Scaling real-time Event Detection to Massive Streams

Dominik Wurzer

Doctor of Philosophy
Institute for Language, Cognition and Computation
School of Informatics
University of Edinburgh
2017

Abstract

In today's world the internet and social media are omnipresent and information is accessible to everyone. This shifted the advantage from those who have access to information to those who do so first. Identifying new events as they emerge is of substantial value to financial institutions who consider realtime information in their decision making processes, as well as for journalists that report about breaking news and governmental agencies that collect information and respond to emergencies. First Story Detection is the task of identifying those documents in a stream of documents that talk about new events first. This seemingly simple task is non-trivial as the computational effort increases with every processed document.

Standard approaches to solve First Story Detection determine a document's novelty by comparing it to previously seen documents. This results in the highest reported accuracy but even the currently fastest system only scales to 10% of the Twitter stream. In this thesis, we propose a new algorithm family, called memory-based methods, able to scale to the full Twitter stream on a single core. Our memory-based method computes a document's novelty up to two orders of magnitude faster than state-of-the-art systems without sacrificing accuracy.

This thesis additional provides original work on the impact of processing unbounded data streams on detection accuracy. Our experiments reveal for the first time that the novelty scores of state-of-the-art comparison based and memory-based methods decay over time. We show how to counteract the discovered novelty decay and increase detection accuracy. Additionally, we show that memory-based methods are applicable beyond First Story Detection by building the first real time rumour detection system on social media streams.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Dominik Wurzer*)

Acknowledgements

First and most of all, I want to thank my supervisor Victor Lavrenko for his outstanding support, help, patience, motivation and guidance. I will miss the meetings and discussions, which enabled me to learn from the greatest teacher I have come across.

I also want to thank Henry Thompson and Miles Osborne for their support and discussions.

Table of Contents

Chapter 1

Introduction

Historically, most information was reserved for an elite group who had exclusive access. The advantage was with those who were granted access to the information. With the emergence of news organizations, these barriers were opened by allowing everyone to access information about events happening around the world. Until the end of the last century, information was reported and spread by news agencies. Their professional journalists carefully researched and curated news stories which resulted in a low volume of high-quality information. Although free media made information more accessible, they still controlled it by deciding what information was of interest to the public.

The internet and the free accessibility of data from social media services act as an equalizer that provides access to information to everyone. In particular the emergence of social media decentralized news, as it provides everyone the means to report and share information at ease.

Information shared on social media channels differ from official news-wire stories. Their messages are shared publicly although they are often targeted at a specific and often small audience to which they are relevant. Consequently, the vast majority of stories on social media services like Twitter can be regarded as irrelevant to the general public as they consist of trivial or private chatter. Although noisy, the realtime nature of these fast-moving data streams allows gathering an insight on what is going on at the moment, which makes them a valuable information source. This information is freely accessible to everyone. The benefit no longer resides with those who have access to it but with those who do so *first*. The identification of novel information in realtime is challenged by the sheer volume and unorganized nature of the available data.

This becomes particularly apparent in economics. The efficient market hypothesis by Nobel Prize winner Fama (1969) states that it is impossible to consistently beat the markets on average profit return if all information is equally distributed. In a market where information is uniformly available to all participants, trading options cannot be recognised as over- or undervalued. Consequently, continued profitability above the market average is not possible as the success only depends on external unknowns and coincidences. According to the efficient market hypothesis, continued economic success can only result from having an information advantage over competitors. This explains the interest in information gathering and processing from financial institutions. The earlier and faster information is acquired the greater advantage it provides over competitors. Consequently, the value of information is determined by the delay before it can be acquired.

Detection and analysis of information in social media streams are also in the interest of governments. The gathering of realtime information allows for better and faster management of governmental responses to events like earthquakes, terror attacks and incidences or mass panics surrounding big events like the Olympics or festivals (Hosenball, 2011). Fast detection of new information improves modelling rapidly evolving situations. By analysing public data streams in real-time, governmental agencies are able to reduce response latencies and allocate resources more efficiently. As in economics, the value of information depends on the delay before it can be detected. The earlier new information is detected, the lower the response times and the bigger their impacts are.

Detecting new information about evolving events in realtime is a valuable but also very challenging task. Social media services produce several thousand messages every second. This clearly surpasses human capacities, so automatic processing and extraction of information about events from such fast-moving and noisy streams would clearly provide an advantage. The Topic Detection and Tracking (TDT) program[1], which started in 1998, aimed to create automated systems that assist human analysts in monitoring news streams around the world. Among several subtasks, TDT introduced

[1]TDT by NIST - 1998-2004. http://www.itl.nist.gov/iad/mig/tests/tdt/resources.html (Last Update: 2008)

the task of automatically detecting new events in data streams. TDT vaguely defines events as something that happens at a specific time and place. New Event Detection, also called First Story Detection (FSD) describes the task of monitoring a stream of documents with the intent of identifying those that speak of unseen events, with respect to all previously seen documents, first. While FSD technology was originally designed to operate on small scaled news streams, state-of-the-art systems process millions of documents. Unfortunately, even the fastest FSD systems fall short, compared to the volume of today's social media streams. This is problematic because for many applications the value of information gathered by an FSD system depends on the delay before it can be detected. To reach its full potential, an FSD system has to be able to scale to the full size of the stream it operates on.

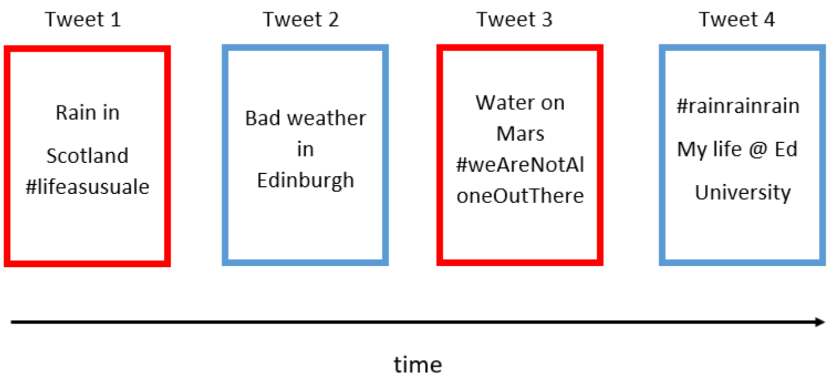

Tweet 1
Tweet 2
Tweet 3
Tweet 4
Rain in Scotland #lifeasusuale
Bad weather in Edinburgh
Water on Mars #weAreNotAloneOutThere
#rainrainrain My life @ Ed University
time

Figure 1.1: Example of a tweet stream containing 4 documents (tweet 1 - tweet 4). Tweet 1 and Tweet 3 are considered "first stories", since they report about an event first. Tweet 2 and tweet 4 are "follow-up" stories.

The challenge we tackle in this thesis is detecting novel events in streams of text as fast as possible - preferably instantly - after their publication. This task is called "New Event Detection" or "First Story Detection" (FSD). Figure 1.1 shows an example of a tweet stream containing 4 units (tweets). In a streaming setting, units arrive one at a time in a defined order, often chronological. In this thesis, we refer to all stream units, like tweets and news articles, as documents. FSD systems are required to processes documents and determine whether they speak about previously unseen events. Documents like *tweet 1* and *tweet 3*, reveal a high degree of novelty with respect to the past. Consequently, these documents are likely to talk about new events. We refer to them as "*first stories*" because they talk about new events first. Documents that share topicality with previously arrived documents, like *tweet 2* and *tweet 4*, are considered as “follow-ups”. FSD requires distinguishing “first stories“ from “follow-ups”. The main challenges of FSD arise from the ever growing number of tweets processed, which determine the degree of novelty of future documents and the high frequency in which they arrive.

FSD systems are applied to high-volume streams, which move too quickly to be processed manually. Modern data sources like Twitter produce thousands of documents every second. Such high-volume streams require operation with high efficiency. For FSD systems, the term efficiency addresses two concepts: 1) *throughput*, the number of documents processed per time unit, and 2) *latency*, also called *lag*, the time that passes until a decision is made. Both, high throughput and low latency, are necessary to successfully operate an FSD system on a high-volume stream.

The only published algorithms that can detect novelty without any lag are comparison based methods. This family of algorithms computes document novelty based on vector proximity with previously encountered document representations. The degree of novelty is defined by 1 minus the similarity to the closest previous document representation. In addition to comparison based methods, novelty can also be detected by counting based approaches. These describe another family of algorithms, which is also referred to as trend-based novelty detection. Trend based algorithms observe term and phrase counts over time and focus on detecting “burstiness” behaviour in them, which is assumed to signal new topics. New topics are likely discussed using new terms or a new combination of terms. Consequently, changes in the frequency terms and phrase usage are interpreted as signals for new events. While effective for the identification of trending topics in social streams, this algorithm family is unfortunately unsuitable for First Story Detection for two reasons:

1) Delayed Detection:
Trend based novelty detection is limited to retrospective detection. Repeated signals are necessary to alter term statistics sufficiently enough to detect an event. FSD however, requires instant detection and does not allow any lag.

2) Limited to large events:
Trend based novelty detection can only detect events large enough to alter the term

statistics. This means that trend based novelty detection are unable to detect small events that only span of a few messages.

We conclude that trend based methods are unsuitable for our task because novelty detection is delayed and limited to large-scale events.

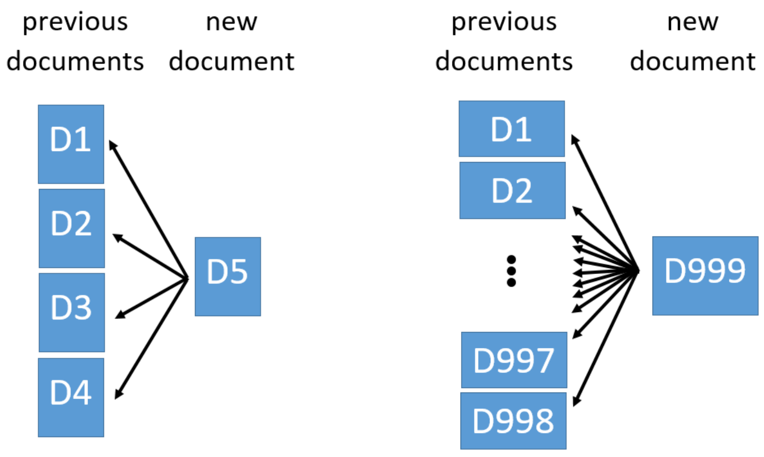


Figure 1.2: Principle of comparison based novelty computation: New documents are compared to all previously arrived documents to identify the most similar one. Each processed document increases the computational effort of future documents.

State-of-the-art FSD systems apply comparison based novelty detection because it results in the highest accuracy and is the only published algorithm family that introduces no lag. The downside of comparison based novelty detection is that the space and time complexities depend on the number of documents processed in the past. Figure 1.2 illustrates that comparison based methods require $n-1$ distance computations (comparisons), to determine the novelty of the n^{th} document arriving from the stream. The number of terms two documents share in common determines the number of scalar multiplications necessary to compute their distance. The computational cost is manageable when the number of documents remains low. On data streams however, the number of previously seen documents increases without bound.

This is challenging because new documents are compared to a continuously growing number of documents. Consequently, comparison based novelty detection becomes progressively slower until it gets prohibitively expensive. In addition to a steadily increasing processing time, comparison based methods also suffer from a steadily increasing storage cost, as each new document needs to be remembered for future comparisons.

High-volume streams in combination with ever increasing document counts demand efficient operation in constant time and space on a per document basis. Efficiency increases when the computational effort on a per document basis is reduced. To scale to fast-moving data streams, modern comparison based FSD approaches limit the number of comparisons made by narrowing the search space. This sacrifices guarantees on errorless operation to reach an adequate level of efficiency. This leads to gains in efficiency at the cost of effectiveness. Although state-of-the-art FSD systems only grow marginally in space over time, they eventually run out of space and turn infeasible. Additionally to the established streaming requirements, including operation in constant time and space, we introduce another requirement: operation with constant accuracy over time. Constant accuracy over time is an overlooked constraint on streaming algorithms, but we argue that this requirement is as important as operation in constant time and space. If performance degrades over time, an FSD system would at some point become impractical when applied to an unbounded data stream. To operate in a true streaming environment, a system must perform highly efficiently while guaranteeing constant accuracy in constant time and space.

Based on the above discussion we define key elements on how this thesis contributes to the improvement of FSD. Addressing the downsides of comparison based novelty detection, we introduce a new algorithm family for novelty computation, called memory-based methods. In contrast to comparison based methods, memory-based novelty computation does *not* compute novelty based on document-level comparisons. Instead, a combined memory - covering all previously encountered information - is constructed and used to detect unseen information based on term-level comparisons. We show that memory-based methods overcome the disadvantages of comparison based methods while maintaining their high effectiveness and operation with zero lag.

Furthermore, we study the impact of unbounded data streams on FSD accuracy over time. Our experiments show that accuracy decreases for comparison and memory-based algorithms and provides counter measures. Additionally, we show that memory-based methods are applicable beyond novelty detection for FSD. We the first real-time rumour detection system by computing features based on information entailment using memory-based novelty detection.

Central Hypothesis

---

memory-based algorithms detect new events orders of magnitude faster than comparison based approaches while achieving the same level of accuracy.

Novelty Computation

State-of-the-art novelty computation relies on computing similarities with previously seen documents. This imposes a limit on how far FSD can be scaled up without losing effectiveness. In this thesis, we introduce memory-based novelty detection, the first effective non-comparative algorithm family for novelty detection with zero lag. Instead of pairing documents and computing similarities between them, we construct a single representation of the past, covering information from all previously encountered documents. Novelty is computed by measuring a document's overlap with the past. Our experiments show that non-comparative memory-based novelty computation reaches state-of-the-art FSD effectiveness that is statistically indistinguishable from comparison based methods.

Efficiency

The efficiency of FSD systems addresses two properties: lag and throughput. FSD systems should be efficient enough to scale to real life high-volume data streams without sacrificing effectiveness. This requires throughput greater than the stream volume in conjunction with zero lag. We show that memory-based novelty computation is superior to comparative approaches in terms of throughput while both methods achieve zero lag. Shifting from document level comparison to comparison with a single representation of the past exceeds the throughput of state-of-the-art FSD approaches by 1-2 orders of magnitude. We show how memory-based FSD can process the equivalent of the full Twitter stream using only a single core.

True streaming environment
Streams are by definition infinite data sources. In order to operate in a streaming environment, an FSD system is required to operate in O(1) (constant) in terms of time and space with respect to the number of documents processed. We reveal that even the most effective FSD approaches cannot operate in constant time and space. In contrast, we show that memory-based FSD operates in absolutely constant time and space, allowing it to operate in a real streaming environment.

In addition to restricting time and space, we require FSD systems to keep the detection accuracy constant over time. We provide the first study of FSD accuracy over time and reveal that the performance of comparative and non-comparative FSD approaches degrades over time. We argue that constraining accuracy is as important as time and space to retain an algorithm's usability when operating on unbounded data sources. Our experiments reveal that the performance decay can be modelled using a simple mathematical decay function. Counteracting novelty decay over time results in significantly increased detection performance. Additionally, we show that traditional speed-up techniques for FSD unwittingly prevent novelty decay over time.

The main contributions of this thesis are:

- We show that a document's degree of novelty can be computed effectively and efficiently without computing similarities to other documents
- We present a method that solves FSD in constant time/space
- We present a method that solves FSD much faster than in the past (10-200x)
- We show that considering recency improves comparative and non-comparative FSD effectiveness
- We show that the novelty scores of comparative and non-comparative FSD system decay over time and provide a method to counteract novelty decay
- We show that novelty detection can be used to significantly improve rumour detection accuracy

Chapter 2

Background

The following chapter reviews the research fields in computer science necessary for understanding the rest of the thesis. Initially, we introduce the Topic Detection and Tracking initiative, which invented the core task of this thesis, First Story Detection. Because First Story Detection is a streaming task, we review the implication of unbounded data sources and explore approaches to dealing with every growing numbers of documents. The main constituent of this chapter explores and compares existing approaches to solve the First Story Detection task. Eventually, this chapter reviews the evaluation paradigms used in the literature to evaluate First Story Detection performance.

2.1 Topic Detection and Tracking

Topic Detection and Tracking (TDT) is a research body and evaluation paradigm for the organization of news streams. The growing volume of information published by broadcasters worldwide exceeds human capabilities in detecting and organizing relevant information within the stream. The TDT program aimed to assist human agents, monitoring broadcasting streams, by automatically organizing event based information. The U.S. government funded the TDT program, which started with as a pilot study in 1997 and ran for 5 years. During the following years, 4 competitive system evaluations took place, which focused on improving detection accuracy (Fiscus & Doddington, 2002). The goal of these competitions was to intensify research and build technology aimed at automatically detecting and monitoring topics in broadcast streams. The data streams, provided by the Linguistic Data Consortium[1], covered

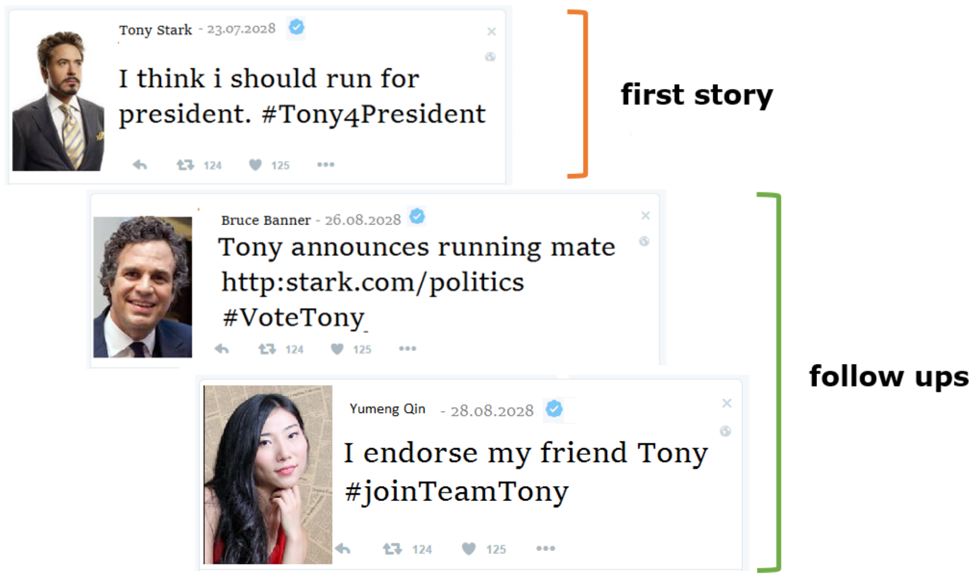
[1]https://catalog.ldc.upenn.edu/LDC98T25

news-wire articles as well as speech-to-text transcripts of television and audio podcasts in Chinese and English.

In news streams, topics suddenly emerge, become popular and eventually fade away. Relevance is therefore bound by a temporal component. Information currently relevant to a human analyst might be irrelevant at a later point in time. In contrast to traditional Information Retrieval tasks, TDT does not rely on the concept of relevance but introduced a topic-based organization instead. Initially, topics were vaguely defined by a notion of "aboutness" and "something that happens at a particular time and place" (Allan, 2002). This definition was expanded during the second year of TDT to: "a seminal event or activity, along with all directly related events and activities" (Fiscus & Doddington, 2002). The final definition of a topic starts with its first occurrence in a document and covers all subsequent stories that share the same aboutness. TDT relied on human judgement to decide whether documents share the topicality of target topics. This imprecise definition of what topics are led to studies exploring varying granularities of topics (Allan, 2002) and the hierarchical representations of different levels of topic granularity (Zeng et al., 2009;Kwak et al., 2010).

In this thesis, we use public data sets that follow the original TDT definition of what topics are. These data sets were created by human annotators who defined a set of target topics and decided about the topicality of documents belonging to them using their intuitive human judgement. Documents associated with target topics are not further organized into sub-events or ordered by hierarchical representations. For the remainder of this thesis, we rely on the judgement of the annotators who crafted the data sets and are not further concerned with topic granularity or alternative topic definitions. As in TDT, we assume that each data set has a defined set of target topics. Every target topic comes with a precisely defined starting point, marked by a trigger event. The trigger event is determined by a "first story" - the document that speaks about the topic first. All subsequent documents, reporting about further development of the topic, are dubbed "follow-ups". These documents were identified by human annotators, who considered them to share the same topicality as their corresponding target topic.

Figure 2.1 shows an example topic called "presidential campaign of Tony Stark". The first document in the stream stating that Tony Stark declares running for the presidency is the trigger event, also called the "first story". All subsequent documents


Tony Stark - 23.07.2028

I think i should run for president. #Tony4President

124 125

Bruce Banner - 26.08.2028

Tony announces running mate http:stark.com/politics #VoteTony

124 125

Yumeng Qin - 28.08.2028

I endorse my friend Tony #joinTeamTony

124 125

Figure 2.1: Example of a "first story", marking the trigger event of a topic and its corresponding "follow-ups" - subsequent documents that share the same topicality.

talking about other's reactions, the announcement of the running mate, fund-raising and campaigning, are part of the same topic and thus considered as "follow-ups". After the election, the topic would fade and eventually vanish.

TDT Data Sets

The TDT program focused on organizing news stories. Broadcasters continuously publish news stories, which naturally form a stream of information. Formally, streams are unbounded data sources. This means that documents continuously arrive one-at-a-time in chronological order defined by their publication time stamp. In the course of the TDT program, the Linguistic Data Consortium[2] contributed 3 independent streaming data sets. They consist of a collection of news-wire articles, radio and television material from 1994 – 1998 in the form of text and speech in English and Mandarin. All documents are organized chronologically by their publication time stamp to form a data stream. Audio data was automatically transcribed and Mandarin automatically translated into English. The three resulting corpora vary in size and range from 26,000 to 74,000 documents. The number of documents appears to be low by today's standards but was considered to be reasonable in the early 2000s. Note that the TDT program exclusively focused on detection accuracy, neglecting efficiency and scale. Modern data sources require a combination of accuracy and scalability to keep FSD

[2]https://catalog.ldc.upenn.edu/LDC98T25

feasible. In this, thesis we base our evaluation on novel and large-scale social media data streams that reflect true streaming environments.

2.1.1 TDT Sub-Tasks

The broader problem of event based news organisation was split up, forming 5 different sub-tasks, each of which covers an intrinsic part of a news organisation and monitoring system:

1. **Story Segmentation**
   Broadcast shows on TV usually cover a number of different stories within a single show. TDT systems process these shows in the form of automatically transcribed text streams. While differentiating between different stories does not impose a challenge to humans, it is not clear for a machine where in an automatically transcribed stream of text one story starts and the other one ends. The goal of Story Segmentation is to detect the boundaries, segmenting different news topics within a continuous stream of text (Allan, 2002). Story Segmentation was considered as a pre-processing step for subsequent TDT tasks (Fiscus et al., 2002).

2. **Cluster Detection**
   Cluster Detection describes the task of grouping documents by their topicality. Each resulting cluster corresponds to a topic within the stream. Cluster Detection is unsupervised as neither the topics themselves, nor the number of topics are known in advance. The streaming nature of TDT requires online clustering. New topics trigger the creation of new clusters and follow-up stories are assigned to their corresponding clusters.

3. **First Story Detection**
   First Story Detection (FSD) identifies new events in a stream of documents. More formally, FSD describes the task of identifying those documents that speak about previously unknown events first. Allan (2002) referred to FSD as a special case of Cluster Detection that focuses on the problem of when to spawn a new cluster. First stories are detected by computing a document's novelty with respect to the past. Since novelty refers to the past, the degree of novelty of a particular document depends on the point in the stream where the FSD system was turned on. FSD is considered to be the hardest of the 5 TDT tasks

(Petrovic, 2013). In TDT, First Story Detection instantly detects new events on data streams, which provides direct applications for news, financial and security institutions (Allan, 2002).

4. **Topic Tracking**
   Topic Tracking systems are presented with a predefined target topic and a stream of documents. The target topic is defined by a small set of 1 to 4 documents that share the same topicality. Topic Tracking describes the supervised task of identifying all those documents in the stream that share the same topicality as the target topic. Since topics are independent of each other, a document can be relevant to more than one target topic. Topic Tracking, although similar to information filtering, sets itself apart by the absence of feedback about previous decisions (Allan, 2002). Interestingly, Topic Tracking is related to First Story Detection, because Topic Tracking technology can be used to solve FSD and comparison based FSD can solve Topic Tracking (Allan et al., 2000a). In TDT, Topic Tracking allows following and monitoring the development of previously detected events.

5. **Story Link Detection** In Story Link Detection, a system is provided with two different stories and asked whether they share the same topicality. Story Link Detection was considered as a separate sub-task because it represents a core technology of TDT – the measurement of document similarity. A system's performance in Story Link Detection directly carries over to comparison based FSD and Topic Tracking.

The 5 sub-tasks assemble a complete TDT system that identifies new topics and follows them as they evolve.

2.2 Streaming Environments

Data streams, although omnipresent in modern life, are often not recognized by those who engage with them on a daily basis. The general public considers "streaming" as on-the-fly data processing, like streaming videos from servers to laptops or TVs.
The meaning of "streaming" in computer science and IR in particular, differs from the general perception of the term. In computer science, streams are considered to be unbounded data sources, where new items arrive on a continual basis one-at-a-time in a

particular order. Real world examples of such data streams are passenger IDs traveling through a public transport system, photos and videos uploaded to the internet, trade orders arriving at stock exchanges, posts on social media services and news articles published by broadcasters.

In the thesis we refer to data streams as unbounded data sources where documents arrive in chronological order - defined by their publication time stamp, one at a time. This matches the definition of streaming environments by Muthukrishnan (2005). Data streams differ from batch data sources as they have neither a beginning nor an end. Documents are organised in time and can only be accessed sequentially in the order they arrive.

The public availability and ease of accessibility of data streams from social media services have recently sparked broad interest in various research fields. Twitter's document stream for example, was used for natural disaster detection (Sakaki et al., 2010), rumour detection (Qazvinian et al., 2011), disease outbreaks prediction and tracking (Achrekar et al., 2011), monitoring the aftermath of terror attacks (Agrawal et al., 2010), mining the opinions on products (Jiang et al., 2011), stock market prediction (Bollen et al., 2011) flu outbreak predictions (Ritterman et al., 2009), election outcome predictions (Sang et al., 2012) and building trading strategies (Sul et al., 2014), to name a few.

2.2.1 Implication of Unbounded Data Sources

Unbounded data sources differ from finite batch data because documents arrive on a continual basis. Storing an ever growing amount of data would become prohibitively expensive, which requires algorithms to perform computations on-the-fly (online). In this thesis, we assume operation on high-volume data streams. This means that we expect to receive several thousand documents every second. When documents arrive at such a high rate, algorithms are required to process them instantly. High-volume streams only allow for a single pass over the data, as there is no time to revisit a document at a later point in time. The algorithmic complexity of streaming algorithms determines the amount of resources (time/space) necessary to process a particular item in the stream. Muthukrishnan (2005) formulated desiderata for streaming algorithms that require operation in O(1) with respect to processing time and memory footprint on a per-document-basis. This requires constant processing time and space for each document, independently from its position in the stream. Note that this resembles the strongest possible condition for algorithm complexity. Operation in constant time and space allow algorithms to remain feasible as they see more and more data. By contrast, algorithm with time and space complexities dependent on the number of documents processed become prohibitively expensive. (Qin et al., 2017).

In this thesis, we expand the desiderata for streaming requirements by Muthukrishnan. In addition to operation in constant time and space, we also require an algorithm's accuracy to remain constant over time. The impact of changing effectiveness over time is an overlooked topic in streaming applications and research, which we find to be as significant as the operation in constant time and space. If a system's accuracy were to degrade as it processes more and more data, it would eventually turn impractical.

Based on the discussion above, we require an FSD system to operate in a true streaming environment. This requires performing all operations in constant time and space while also maintaining a constant detection accuracy over time.

2.2.2 Dealing with Unbounded Data Sources

The recent emergence of public social media streams has sparked a growing research interest in processing them. Various approaches have been proposed to scale algorithms to unbounded data sources. Unfortunately, many researchers confuse data streams with collections containing large amounts of documents. They consequently focus on improving efficiency, while neglecting operation in constant time and space. We only consider solutions that are applicable in a true streaming environment.

All approaches to deal with data streams share a common principle. In a nutshell, constant operation is achieved by limiting the amount of data presented to an algorithm (Qin et al., 2017). We conclude that constant operation can be reached when an algorithm is exposed to a constant amount of data. The remainder of the section summarizes different approaches to keep an algorithm's data exposure constant.

The simplest solution on scaling algorithms relies on sliding windows (Datar et al., 2002). With each arrival of a new document, an old document falls out of scope. The number of documents stored equates to the length of the window, which moves through the stream. The fixed size of the window guarantees a constant data exposure to the algorithms.

While time-sensitive windows are widely applied due to their simplicity, other deletion strategies can provide application dependent advantages (Lee et al., 2014). They conclude that sliding windows are effective whenever the application dependent focus can be captured by the deletion criteria. Examples include: the deletion by recency, whenever newer items are believed to carry more information than old ones (Li et al., 2008); or deletion by frequency, when the most frequently observed items are deemed more important than less frequently observed items (Berinde et al., 2009).

Sampling strategies provide an effective alternative to window based strategies (Braverman et al., 2012). Sampling techniques select which items of the past are to be used for future computations. Ideally, a sample captures all relevant characteristics of seen data, thus providing a compressed model of the past (Shrimpton et al., 2015). Various sampling techniques for data streams have been proposed, providing uniformly random (Vitter, 1985) as well as biased (Aggarwal et al., 2006; Osborne et al., 2014) models of the past. Sampling differs from sliding windows as it provides certain guarantees on the data selected and not every new item is ensured to be added to the sample. Constant data exposure guarantees operation in constant time and space. Unfortunately, selective data removal inevitably comes at the cost of information loss. Therefore, all discussed methods ensure high efficiency and scalability at the expense of effectiveness.

2.2.3 Data Streams in First Story Detection

When TDT introduced the First Story Detection task, it was advocated as a streaming task (Allan, 2002). TDT, however, chose not to impose the strong conditions on algorithm complexity necessary for true streaming applications. Instead, TDT only enforced online processing while focusing on effectiveness and neglecting efficiency and scale. The resulting algorithms (Allan et al., 2000) are state-of-the-art in terms of detection accuracy but at the same time inapplicable in a true streaming scenario. The emergence of massive social media streams highlighted the scaling issues of systems designed for the TDT competitions.

Social media and news-wire streams

Social media differs from traditional news-wire streams, which imposes additional challenges on FSD systems. The most obvious difference is the massive volume of social media streams, which exceeds those of broadcasters by several orders of magnitude (Petrovic et al., 2013; Osborne et al., 2014). Microblogging services like Twitter[3] or Sina Weibo[4] produce several thousand messages every second, requiring highly scalable algorithms.

Apart from scale, social media streams are not primarily used for the distribution of "news worthy" information. On a broadcast stream, all documents represent information carefully curated and deemed to be relevant to the general public by professional editors. Content in social media streams, on the contrary, is produced by ordinary users and mainly consist of information relevant to a specific often small group of individuals. This covers chatter and trivial conversation about non-newsworthy topics. We refer to this chatter as noise that needs to be separated from the signal (news worthy information), because it would distract a human analyst from identifying significant events.

Social media streams also differ in the usage of language. Social media represents an informal communication channel for ordinary users. Messages reveal a highly creative use of language and lexical variations. Users tend to create new and compound terms (hashtags) and change or bend the meaning of existing terms. This aspect of social media streams is of great interest for the study of the evolution of human language but also introduces vocabulary mismatches, which challenge Information Retrieval approaches (Petrovic, 2013).

[3]Average Twitter volume 5,787 tweets/second https://about.twitter.com/company (last updated: March 31, 2016)

[4]Average Sina Weibo volume 1,200 weibos/second http://open.weibo.com/ (last updated: July 31, 2016)

2.3 Approaches to First Story Detection

Over the years various approaches including probabilistic, IR and classification-based approaches have been proposed to tackle First Story Detection with varying success. The most successful systems today rely on information retrieval techniques, which is why we discuss them in more detail. The core component of an FSD system is its novelty computation procedure. We distinguish between comparative and non-comparative novelty computation.

2.3.1 Principle of Comparison-based Novelty Detection

FSD detects a new event by identifying its first story - the document that mentions it first. The core hypothesis of comparison based novelty detection assumes that first stories are different from all previous documents because they contain unseen information – the new event. Detecting a new event is then subject to measuring how different a document is from those seen in the past. For comparison based novelty detection, a document's (d_t) degree of novelty is determined by the distance to the closest previous documents $(d_1...d_t)$.

$$Novelty(d_t) = 1 - \max_{d \in d_{1...d_t}} similarity_t(d_t, d) \tag{2.1}$$

Identifying the closest document is also referred to as nearest neighbour search. Comparison based systems set themselves apart in the way they solve the nearest neighbour search problem.

The most successful FSD systems, including the UMass system (Allan et al., 2000) and CMU system (Yang et al., 1998) apply the vector space model (VSM) by Salton et al. (1975). The VSM describes a multidimensional space, whose axes correspond to unique terms in the vocabulary. Let us denote by $t_{d,n}$ term t in document d_n from all documents D. The dimensionality of the vector space equates to $Dim = |\{t : t \in d, d \in D\}|$

In the VSM, all documents are represented by vectors in the term space. This allows approximating document similarity through vector proximity. Document vectors are usually weighted by the $tf.idf$ scheme, a combination of term frequency and inverse document frequency (Salton et al., 1975).

$$weight(t,d) = tf(t,d) * idf(t) \tag{2.2}$$

The term frequency $tf(t,d)$ represents the number of times term t occurs in document d. High term frequencies indicate higher representativeness of the term t for the document d. The inverse document frequency $idf(t)$ is an indication of the overall importance of term t. Terms that appear in many documents are less representative for an individual document (Salton et al., 1975). The inverse document frequency (idf) of term t is defined by the ratio of all documents to the number of documents term t appears in.

$$idf(t) = log(\frac{|D|}{|\{d : t \in d, d \in D\}|}) \tag{2.3}$$

Since FSD systems operate on data streams, the idf component can either be computed incrementally or on a separate corpus (Yang et al., 1998).

The combination of the local (document-based) and the inverse of the global (collection-based) document frequency indicate a term's degree of representativeness for a document.

The pilot study of TDT by Allan et al. (1998) explored various ways of measuring document distances and their impact on FSD performance and found tf.idf weighted cosine similarity to be superior. Cosine similarity measures the similarity of two documents (d_1, d_2) by the angle between their corresponding vectors.

$$\text{cosine similarity}(d_1, d_2) = \frac{\sum_{t \in d_1 \bigcap t \in d_2} tf.idf(t, d_1) * tf.idf(t, d2)}{\sqrt{\sum_{t \in d_1} tf.idf(t, d_1)^2} * \sqrt{\sum_{t \in d_2} tf.idf(t, d_2)^2}} \tag{2.4}$$

The cosine similarity can be interpreted as a judgement of vector alignment. Resulting from the projection of two unit vectors, the cosine similarity is naturally bound between 0 (orthogonal direction) and 1 (same direction), independently from the number of dimensions.

2.3.2 Comparison based FSD Systems

The performance of comparison based FSD systems depends on the quality of their document similarity measurement. We review 3 comparison based FSD systems, the Dragon system (Yamron et al., 1998), the UMass system (Allan et al., 2000) and the CMU system (Yang et al., 1998), which are known for their high FSD performance during the TDT competitions (Petrovic, 2012). All 3 systems rely on the cosine similarity (Equation 2.4) for their novelty computations, which was found to result in the highest accuracy during the initial TDT workshop (Allan, 2002). The three systems set themselves apart in the way they organized previously received documents.

UMass
The principle of the UMass FSD systems (Allan et al., 2000) is considered to be the standard approach to FSD (Petrovic et al., 2010; Wurzer et al., 2015). For each document arriving from the stream, UMass computes its tf.idf weighted cosine similarity with all previously arrived documents. The novelty score of a document equates to 1 minus the maximum similarity with a previous document, as seen in Equation 2.1. The exhaustive comparison with all previous documents is computationally expensive but results in state-of-the-art FSD accuracy. The worst case time and space complexity of UMass depends on the number of documents (n) processed , resulting in $O(n)$. This means that the time and space complexity increases linearly with the number of documents processed. To speed up computation, UMass applies an inverted index and limits document vectors to the t highest weighted features. This reduces the average time complexity from $A(n*dl)$ to $A(n*dl/v)$ operations. Here, dl refers to the average document length, n is the number of documents processed in the past and v is the number of unique terms found in the n documents.

Unfortunately, UMass does not scale to data streams because its time complexity depends on the number of documents encountered in the past. Consequently, the runtime per document and overall storage cost grow without bound. We were provided with the source code of UMass and use it as a high effectiveness baseline due to its state-of-the-art FSD accuracy. An in-depths analysis of the UMass FSD system appears Chapter 3.

Dragon

Algorithm 1 : Dragon FSD System

```
for all $document\ d \in Stream$ do
    for all $term\ t \in d$ do
        for all $centroid\ c : t \in c$ do
            update cosine similarity$(d,c)$
        end
    end
    $sim_{max} \leftarrow max_{c'}\{$cosine similarity$(d,c)\}$
    noveltyScore(d) $\leftarrow (1 - sim_{max})$
    if $sim_{max} < \alpha$ then
        for all $term\ t \in d$ do
            update(c',t)
        end
        discard(d)
    else
        $c_{new} \leftarrow d$
        discard existing cluster
    end
end
```

Unlike UMass, the Dragon system (Yamron et al., 1998) does not represent the past by individual documents. Instead, documents arriving from the stream are grouped by their topicality using single pass agglomerative clustering. Algorithm 1 shows the detailed pseudo-code of the Dragon system according to their publication (Yamron et al., 1998) and Figure 2.2 illustrates a simplified 2-dimensional view of Dragon's vector space model. Each cluster is represented by a centroid vector ($C1$ and $C2$), formed by averaging all document vectors ($D1 - D6$) associated with it. New documents ($D7$) arriving from the stream are compared to all centroids using cosine similarity (Equation 2.4). If the distance to the closest centroid vector falls below the integration threshold (α), documents get assigned to it and contribute their statistics to the centroid's vector. The document's information is then captured by its corresponding centroid vector and the Dragon system discards it. The resulting centroid vectors resemble the average of their constituents and the clusters form hyper spherical shapes, as seen in Figure 2.2.

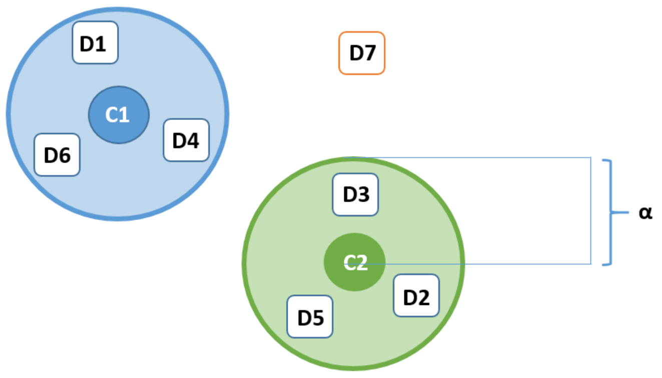


Figure 2.2: Simplified 2-dimensional vector space illustration of Dragon's cluster based approach; showing clusters ($C1$,$C2$) and documents ($D1-D7$) and integration threshold α.

Documents that are not similar enough to any existing cluster, like $D7$, are considered to talk about a new event and subsequently spawn a new cluster, using the document vector as the centroid vector. Yamron et al. (1998) state that at any point there exist exactly k clusters. From their publication it remains unclear how the number of clusters is kept constant as the emergence of new clusters would necessarily trigger the removal of an existing one. Deletion of the oldest or the recent least frequently updated cluster appear to be the most likely deletion approaches.

Representing the past by clusters compresses the information of previous documents to a small set of vectors. Limiting the number of clusters to k results in a space and time complexity of $O(1)$ with respect to the number of documents processed and requires on average $A(k*t)$ operations per document. Here, t denotes the document length and k the number of clusters. Resulting from constant document length t and number of clusters k, the Dragon system meets the scaling requirements for operating in a streaming environment by Muthukrishnan (2005).

Unfortunately, the scalability of the Dragon system comes at a significant cost in accuracy. Yang et al. (1998) compared the Dragon system with UMass and found it to be significantly inferior in terms of detection accuracy. They hypothesised that in order to be effective, an FSD system is required to assess if new documents are sufficiently different from each individual past story instead of their averages. Note that this represents a substantially stronger condition for novelty than that of the Dragon system, which considers new events to be novel with respect to the average of the past. Although scalable, we don't consider the Dragon system as a baseline due to its inferior detection accuracy. Note our requirement in Chapter 1, that FSD systems need to be efficient - scale to modern size data streams while operating with zero lag - without sacrificing accuracy.

CMU

Algorithm 2 : CMU FSD System

```
1: for all document $d \in Stream$ do
2:     initialize $FIFO$
3:     for all term $t \in d$ do
4:         for all document $d' \in \{d' : t \in d', d' \in FIFO\}$ do
5:             update cosine similarity$(d, d')$
6:         end
7:     end
8:     $sim_{max} \leftarrow max_{d'}\{\text{cosine similarity}(d, d')\}$
9:     noveltyScore(d) $\leftarrow (1 - sim_{max})$
10:    if $|FIFO| > k$ then
11:        $FIFO \rightarrow$ oldest $d'$
12:        $FIFO \leftarrow d$
13:    end
14: end
```

The CMU system (Yang et al., 1998) overcomes the scaling limitations of the UMass system while performing more accurately than the Dragon system. Algorithm 2 shows the pseudo code for the CMU FSD system. Instead of limiting the entire past to a constant number of averaged centroids, Yang et al. (1998) hypothesized that the majority of previously encountered documents are irrelevant for current novelty computations. Qualitative analysis of events in the official TDT data streams revealed that topics vanish on average after 2.5 months from the stream. Consequently, the CMU system applies a sliding window and limits previously encountered documents to the most recent 2,500 documents, which represents 2.5 months on the TDT news streams. Resulting from the average number of $A(k*t)$ operation, the time and space complexity of CMU is bound by $O(1)$ with respect to the number of documents processed. Here,

k denotes the window length and t the average document length. Constant operation with respect to time and space allows the CMU system to scale to data streams. Comparison with other systems showed that the CMU system is superior to the clustering approach of the Dragon system, but lacks the accuracy of exhaustive comparison of the UMass system. This is expected as discarding old documents inevitably results in information loss.

Luo et al. (2007) improved the efficiency of the CMU approach by only keeping documents considered to be first stories and limiting comparisons to documents matching highly weighted features. These heuristics achieve impressive speed-ups at significant detection accuracy costs.

The CMU and Luo's FSD system apply a sliding window, spanning 2.5 months and 1 month respectively. Note that these systems were designed to process broadcast streams, which at that time produced 2,500 documents in 2.5 months (Yang et al., 1998). On modern data streams like Twitter, 2,500 documents only equate to the past 0.5 seconds. A window spanning 2.5 months would cover $1.296 * 10^{10}$ tweets. To retain feasibility, a window based approach can only capture time periods spanning a couple of seconds when applied to high-volume streams.

In this thesis, we do not use a sliding window based approach as a baseline for the following two reasons:

1) the CMU revealed lower accuracy compared with UMass (Yang et al., 1998);
2) the sliding window allows FSD systems to operate in constant time and space.

2.3.2.1 Scaling Comparison based FSD

Petrovic et al. (2010) introduced the first scalable FSD approach that reaches accuracy indistinguishable from exhaustive comparison based systems like UMass. They achieve high efficiency without sacrificing accuracy by limiting the search space using Locality Sensitive Hashing (LSH). We denote their algorithm as LSH-FSD. LSH is a randomized algorithm that efficiently approximates nearest neighbours with bound error rates. Figure 2.3 illustrates a simplified 2-dimensional model of the vector space using LSH. In a nutshell, LSH randomly segments the vector space using hyperplanes ($h1 - h3$), forming hyper-polygonal shaped buckets, like $B1$ (colourized). Documents

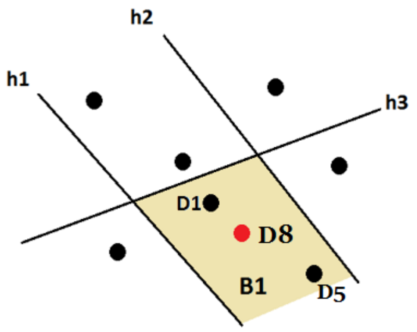


Figure 2.3: The 2-dimensional vector space model illustrates the principle of LSH 3 hyper planes ($h1 - h3$).

are more likely to be similar to documents in the same hash bucket than to documents from other hash buckets. In Figure 2.3, document $D8$ (in red) is hashed into bucket $B1$, which already contains documents $D1$ and $D5$. Instead of comparing to all previous documents, LSH-FSD only compares new documents to those previous documents that fall into the same hash bucket. This allows limiting the search field to only those previous documents that are likely to be similar to new documents arriving from the stream.

LSH by itself does not guarantee constant performance as each new document is stored in its corresponding bucket for future comparisons. LSH-FSD places an upper limit on the number of documents per bucket and removes the oldest document whenever a predefined limit is reached. This is interesting, as deletion occurs whenever a particular region in the vector space becomes too densely populated. The local deletion strategy was found to be superior to temporal based deletion for FSD on Twitter (Petrovic, 2013). Consequently, the time and space complexities are constant (O(1)) with respect to the number of documents processed. The combination of search space reduction through LSH with local deletion drastically increases detection throughput of FSD systems without sacrificing effectiveness. LSH-FSD resembles a state-of-the-art FSD system in terms of efficiency (throughput and lag), effectiveness and scale. Unfortunately, the throughput of the fastest available FSD system still lacks the volume of the full Twitter stream. McCreadie et al. (2013) showed that LSH-FSD requires a cluster of 70 cores to reach a throughput equivalent to those of the average Twitter stream.

In this thesis, we use LSH-FSD as our main baseline because it provides the highest possible efficiency in terms of throughput and lag and reaches performance statistically indistinguishable from exhaustive comparison. We provide and in-depth analysis of the UMass system, whose source code has been made available to us, in Chapter 3

The following two sections briefly summarize non-comparison based approaches to First Story Detection. Although some approaches show promising results, they fail to fulfil our requirements for a true streaming approach to FSD.

2.3.3 Classification based FSD

The highest ever reported accuracy on official TDT data sets was achieved by classification based FSD systems that combine different versions of comparison based FSD systems. The system by Kumaran & Allan (2005) for example, uses a Support Vector Machine based classifier to combines 3 different novelty scores: based on the entire document, named entities and topic terms. Other approaches, like Braun & Kaneshiro (2004) combine several classifiers in a majority voting scheme. Although superior in detection accuracy, classification based FSD systems require labelled training data to learn optimal parameter settings. The annotation process for FSD data sets is labour intensive and therefore expensive. The limited availability of labelled training data limits the applicability of classification based FSD systems.

A conceptually different application of classifiers in FSD is the removal of noise in data streams. Becker et al. (2011) extend the approach of Yang et al. (1998) with a classification step that separates detected events from noise by judging whether they are news worthy.

The most renowned classification based FSD systems is the earthquake detection system by Sakaki et al. (2010). Their system was among the first to successfully scale an event detection system to a social media stream. Contrary to previously discussed FSD systems, classification based FSD systems target specific types of events. Sakaki's system for example, focused on detecting earthquakes and typhoons on Twitter with as little lag as possible.

An initial keyword filter reduces the stream volume to documents containing terms relevant to a target event. Those documents are subsequently classified to be about an actual event using an SVM. If a sufficient number of documents is considered to be about an event within a certain time period, the system concludes that the event is happening. Sakaki's FSD system showed impressive results, reporting earthquakes 6 times faster than official broadcasters. Classification based FSD systems are highly accurate, scalable and detect events with little lag. Unfortunately, classification based systems come with the major drawback of requiring labelled training data, which is expensive to acquire. Additionally, approaches like those by Sakaki lack generality because each type of event requires a separate set of labelled training data and dedicated training. This limits their application to targeted high precision event detection scenarios, where the event types are predefined and training data is available. Since we require a general approach to new event detection, we cannot consider classification based systems.

2.3.4 Trend based Event Detection

On news-wire streams, new topics emerge, grow popular and eventually fade away (Yang et al., 1998). The same applies to social media, where the lifespan of topics tends to be shorter than on news streams (Osborne, 2014). Consequently, documents sharing the same topicality tend to occur in bursts within small temporal proximity (Kontostathis et al., 2003). Trend based FSD, like Cataldi et al. (2010) and Li et al. (2012), monitor the frequencies of terms and phrases with the intent to detect unexpected "burstiness" patterns, which are considered as an indication of new events. Weng et al. (2011) apply wavelet analysis on the frequency-based raw signals of terms occurring in a data stream. They show that this provides a precise measurement of when and how the frequency of individual signals (terms) changes over time (Kaiser, 1994). Weng et al. (2013) detect events by grouping sets of words that describe them with similar burst patterns.

The high-volume of social media streams challenges trend based FSD systems. Cordeiro (2012) tries to reduce the stream volume by observing the usage of hashtags and detecting frequency spikes in them. Trend based FSD systems result in high detection accuracy but they all share a major drawback. New events are recognized by sudden changes in vocabulary usage. Consequently, new events can only be detected once a sufficiently large number of event related mentions have occurred. This only enables detecting events retrospectively which does not fulfil our requirements on FSD. Remember that the value of information depends on the lag before it can be detected. On trend based systems this lag might exceed several hours, which makes them inapplicable for FSD.

2.3.5 Summary of existing approaches to FSD

All discussed approaches to FSD are summarized in Table 2.1. The table compares them with our work based on the requirements of an FSD system, as discussed in chapter 1.

Generality: describes systems able to detect all kinds of events. Systems that only detect a certain kind of events, like crime or natural disaster related events, are not considered to be general.

Accuracy: we consider an FSD system as sufficiently accurate if it reaches state-of-the-art accuracy determined by exhaustive comparison based FSD.

Throughput: we consider an FSD system as sufficiently efficient if it is able to operate on the equivalent of the average Twitter stream (5,787 documents/second)[5].

Constant time/space: we only consider systems constant in time and space if they remain truly unchanged as the number of documents increases.

Instant: systems are considered instant when they judge a document without delay as soon as it is made available.

Unsupervised: a system is unsupervised if it does not require any information, data before and while judging documents.

[5]https://about.twitter.com/company (last updated: March 31, 2016)

Algorithm	**Generality**	**Accuracy**	**Throughput**	**const time**	**const space**	**Instant**	**Unsupervised**
Petrovic et al. (2010) (LSH-FSD)	all kinds of events	state of the art accuracy	$<$ 10% twitter stream (state-of-the-art)	O(1) (YES)	slight growth (NO)	YES	YES
Allan et al. (2000) (UMASS)	all kinds of events	state of the art accuracy	initially high, becomes progressively slower	O(#docs) (NO)	O(n) (NO)	YES	YES
Yang et al. (1998) (CMU)	all kinds of events	below state of the art	high - dependent on window size	O(1) (YES)	O(1) (YES)	YES	YES
Yamron et al. (1998) (Dragon)	all kinds of events	below state of the art	high - dependent on number of clusters	O(1) (YES)	O(1) (YES)	YES	YES
Sakaki et al. (2010)	targeted events (NO)	state of the art	full twitter stream	O(1) (YES)	O(1) (YES)	YES	requires training data
Cordeiro (2012)	high-volume with hashtags	no quantitative evaluation	full twitter stream	O(1) (YES)	growth in hashtags(NO)	5 minute lag	YES
Weng et al. (2011)	high-volume events	high precision	low - dependent #terms	O(#terms) (NO)	O(#terms) (NO)	24 hours lag	YES
Cataldi et al. (2010)	high-volume events	no quantitative evaluation	unclear	O(#terms) (NO)	O(#terms) (NO)	YES	YES
Li et al. (2012)	high-volume events	state of the art	high - dependent on number of clusters	O(1) (YES)	O(#clusters) (NO)	2 hours lag	YES
This work	all kinds of events	state of the art	full twitter stream	O(1) (YES)	O(1) (YES)	YES	YES

Table 2.1: Comparison of FSD approaches based on the requirements of a fully applicable FSD system

System Response	Target	Non-Target
YES	Correct (TP)	False Alarm (FP)
NO	Miss (FN)	Correct (TN)

Table 2.2: FSD evaluation contingency table

2.4 Evaluating First Story Detection

The National Institute of Standards and Technology (NIST) has administered the evaluations of the TDT tasks since 1998. All TDT subtasks are evaluated as detection tasks (Fiscus & Doddington, 2002). For FSD, a system is presented with a stream of documents and the task is to decide for each document whether it is about a new event with respect to all preceding documents. Since FSD is a detection task, first stories are denoted as targets, which are to be detected and follow-ups are denoted as non-targets. Targets can be correctly detected (denoted true positives TP in other IR tasks), or missed (false negatives FN). Falsely detected non-targets are denoted as false alarms (false positive FP), as summarized in the contingency table 2.2.

The detection decision (YES/NO) depends on the novelty score that FSD systems assign to each document. A document's novelty score, usually between 0 and 1, reflects a system's decision about being a target. For example, novelty scores of 1 indicate a high *believe* that a document talks about a new event (target), whereas low novelty scores are assigned to follow-ups (non-targets). The final decision on whether a document is considered to be a target depends on the threshold setting and its novelty score. To maximise performance, the detection threshold is tuned on a separate training set. When a document's novelty score exceeds the detection threshold, it is declared to be about a new event. Documents with novelty scores below the detection threshold are considered to be follow-up stories, talking about a previously detected event.

TDT provides two techniques, a single value metric and a trade-off curve, to represent a system's detection accuracy.

Detection Cost Function

FSD systems aid human analysts in detecting new events in a stream of millions of documents. The task of an FSD system is to reduce the entire stream to only those considered to talk about new events. The Detection Cost Function (C_{Det}) captures the cost of the human analyst when monitoring the outcome of an FSD system (Fiscus & Doddington, 2002).

This single number metric represents detection performance at a particular operation point, defined by the system's settings. Note that the detection cost function C_{Det} is an error metric. This means that lower C_{Det} values indicate higher accuracy. The cost of using an FSD system is based on the miss probability (P_{miss}), which leads to information loss, and false alarm probability (P_{fa}), which unnecessarily increases the human effort. The two probabilities are computed based on the YES/NO decisions assigned to each document, as seen in equation 2.5 and 2.6.

$$P_{miss} = \frac{\text{\#missed targets}}{\#targets} \tag{2.5}$$

$$P_{fa} = \frac{\text{\#false alarms}}{\text{\#non targets}} \tag{2.6}$$

Depending on the application, the costs for missing a new event as well as falsely reporting new events, vary. The Detection Cost Function captures the application dependent costs in their corresponding parameters C_{fa} and C_{miss}. The final formula for the cost function linearly combines miss and false alarm probabilities with their corresponding cost weights, as seen in equation 2.7.

$$C_{DET} = (C_{miss} * P_{miss} * P_{target} + C_{fa} * P_{fa} * (1 - P_{target})) \tag{2.7}$$

Data streams are often noisy, resulting in an imbalance between targets and non-targets. This imbalance is considered by the probability of targets (P_{target}) and non-targets ($1 - P_{target}$). In this thesis, we use the standard TDT settings for the cost function parameters, seen in table 2.3.

To emphasise differences when comparing two high performing FSD systems, TDT normalizes the Detection Cost Function by the performance of a random system. TDT defines random system by either considering all documents as targets (YES) or non-targets (NO), which results from the minimum C_{DET}. In our opinion, this does not

Parameter	Value
C_{miss}	10
C_{fa}	1
P_{target}	0.1

Table 2.3: Standard TDT evaluation parameters

$$(C_{DET})_{norm} = \frac{\overbrace{C_{Det}}^{\text{original detection cost}}}{min(\underbrace{(C_{miss} * 1 * P_{target} + C_{fa} * 0 * (1 - P_{target}))}_{\text{always answer “follow-up”}}, \underbrace{(C_{miss} * 0 * P_{target} + C_{fa} * 1 * (1 - P_{target}))}_{\text{always answer “new-event”}})}$$

resemble a truly random system but rather a deterministic system.

$$(C_{DET})_{norm} = C_{Det}/min(C_{miss} * P_{target}, C_{fa} * (1 - P_{target})) \qquad (2.8)$$

A perfect FSD system would result in $min((C_{Det})_{Norm})$ = 0 and scores equal to 1 indicate random performance. We compute C_{DET} using macro-averaging (Allan et al., 2000), as is usual in TDT, because of its insensitivity to varying topic sizes.

Detection Error Trade-off Plot

The Detection Error Trade-off (DET) curve visualizes the trade-off between misses and false alarms. Figure 2.4 shows an example DET plot comparing the accuracy of three systems with random performance. Contrary to the detection cost, DET plots show performance for the full range of thresholds, providing a more comprehensive picture of a system's detection performance. For example, the plot shows that all three system operate with higher accuracy than random performance. Additionally, the plot shows that system 1 and system 2 result in higher accuracy than system 3 for all possible set-ups. Note that the axes describe miss and false alarm probabilities on normal instead of linear or logarithmic scale. This is unusual but it allows expanding the “high performance” areas of the curve – regions of high precision or high recall. Those two areas are marked by the red circles in Figure 2.4. This allows further conclusion to be drawn from DET plots. For example, system 3 performs better than system 1 and system 2 in high precision and high recall settings. DET plots can additionally show

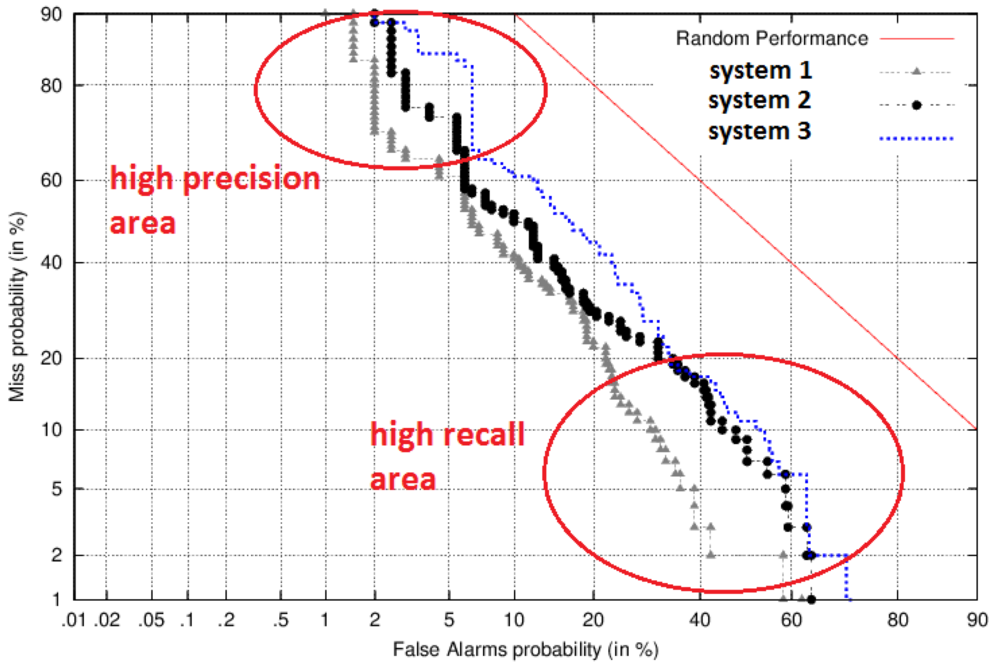


Figure 2.4: Example DET plot comparing the effectiveness of three systems for the full range of parameter settings.

significance corridors, visualizing area of confidence, where a system's performance is indistinguishable from others.

DET plots are constructed by sorting documents for each topic according to their novelty scores and sweeping a threshold through them. At each point in the score space, miss and false alarm probabilities (Equation 2.5 and 2.6) are computed for each topic and averaged (macro-averaging), resulting in the points on the DET plot.

In this thesis, we follow the TDT evaluation paradigm and use the official TDT evaluation scripts (Allan, 2002) using standard parameter settings to compute the normalized minimum detection cost C_{min} and Detection Error Trade-off (DET) plots.

2.5 Conclusion

In this chapter, we presented the background literature that this work builds upon. We introduced TDT - the initiative that invented First Story Detection and the concept of data streams - the data source for First Story Detection. We also explored the implications and countermeasures of unbounded data streams and hinted at our baselines, which are discussed in detail in the next chapter. The chapter also reviewed the evaluation paradigms used to evaluate the performance of FSD systems.

The main section of this chapter was devoted to previous research and approaches to solve the First Story Detection task. We explored the currently most successful approaches, which are based on document level comparisons and identified their weaknesses. Comparison based systems are highly accurate but their processing time and storage cost continuously grows with the number of documents processed. The algorithmic complexity of streaming algorithms describes the resource (time/space) cost with respect to a particular item in the stream. Consequently, the time and space complexity of comparison based systems for the n^{th} document in the stream is $(O(n))$. The most successful approaches overcome these scaling issues using sophisticated document pruning techniques, but their throughput remains too low to process modern size data streams even when deployed on a modern processor.

Chapter 3

Data Set and Baselines

3.1 Motivation

The following chapter outlines the data set upon which our experiments are based on. Additionally, we introduce the two main baselines in great detail.

3.2 Cross Data Set

In this thesis, we make use of the Cross data set, which originated from the Cross project[1], a joint venture between the University of Edinburgh and the University of Glasgow. The project, funded by EPSRC, deals with new event detection on streams such as Twitter, Wikipedia and news-wires. They focused in particular on cross-stream and distributed event detection, for which they also provide a news and a Twitter data set. All Cross data sets resemble document streams, where each document is identified by a unique ID and time-stamp. In this thesis, we make use of Cross's Twitter data set, dubbed Cross-Twitter, for all our experiments on First Story Detection. Cross-Twitter comes in two versions, one covering 52 million, the other 115,000 English tweets from the publicly available 1% sample of the Twitter stream spanning from June 30^{th}, 2011 to September 15^{th}, 2011. Both versions share the same 27 topics and 3,034 annotated documents but differ in the number of unlabelled tweets. The 27 topics range from low volume events (containing only 2 tweets) to high-volume events ($>$ 1.000 tweets) and cover a broad range of events including deaths of celebrities, financial news, natural disasters and scientific discoveries. Most of the events can be found on Wikipedia's

[1]Cross Project: http://demeter.inf.ed.ac.uk/cross/

list of the most important events of 2011[2].

The labelling procedure of Cross-Twitter followed the search guided annotation process, previously applied by TDT, with the difference that the search was limited to 24 hours. This reduces manual labour at the cost of missing a high number of follow-up stories. Missing a substantial number of follow-up stories has implications on FSD performance. We simulate the impact of missing follow-ups by running an identical FSD system (Petrovic et al., 2010) on two data sets, whereas we removed 50% of all follow-up documents in one of them. The resulting FSD performance is illustrated by DET plot in Figure 3.1. The graph reveals higher accuracy when processing fewer follow-ups throughout all threshold settings. This is not unexpected, as a reduced number of follow-ups also reduces the likelihood of mistaking them for first stories (probability of false alarm).

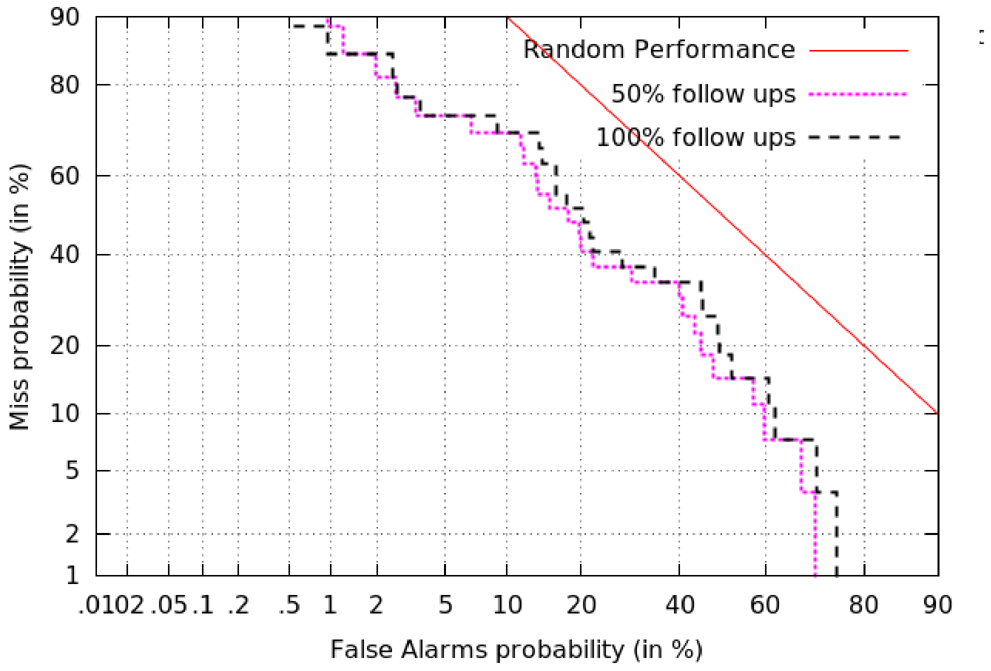


Figure 3.1: DET plot illustrating increased detection performance when removing 50% of the follow-up documents.

Internationally relevant topics like those chosen by Cross, are likely to be discussed by Twitter users within 24 hours after their emergence and afterwards as well. This

[2]Wikipedia 2011 events: https://en.wikipedia.org/wiki/2011

ensures that a reasonable number of follow-ups for all topics in the cross data sets. Limiting the search to 24 hours does not perturb the evaluation process provided that systems are evaluated on the same number of follow-ups.

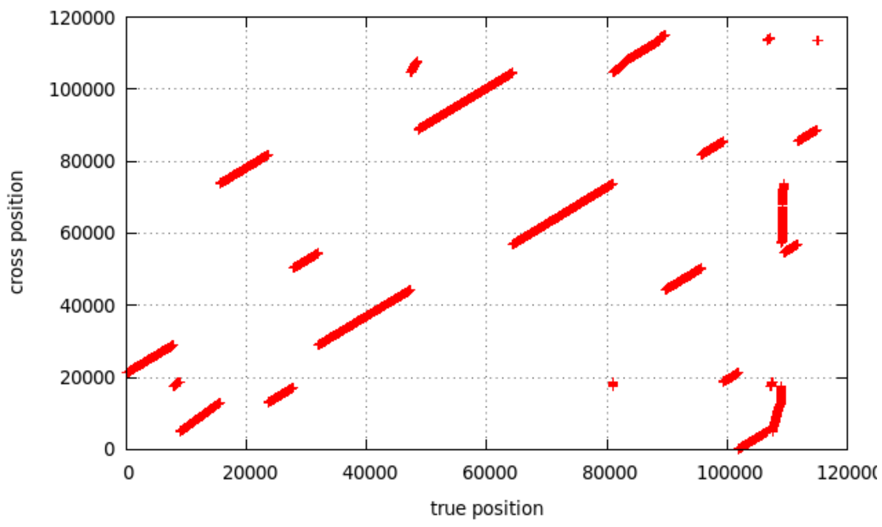


Figure 3.2: Comparing the observed document position in Cross-Twitter 115k with their true position, determined by their publication time stamp.

Cross-Twitter 115k

The 115.000 tweet stream is a sub-sample of the full 52 million tweet data set that includes the same topics and all judged documents. We use the 115k tweets of Cross-Twitter to conduct effectiveness experiments as this allows us to also compare our approach with highly effective but inefficient algorithms.

While exploring the data set, we found a divergence between the observed document position and the order of their publication time stamps, as seen in figure 3.2. The illustration shows that the intra-topic order correctly reflects the time stamp order but the order of the topics themselves has been re-arranged. We could not find information on why the order was changed but hypothesise that topics were re-arranged to be distributed evenly in time.

In a streaming data set documents need to be ordered by their publication time stamp, which reflects the true order in which they occurred. This is particularly important for FSD, where a document's novelty score is based on the documents preceding it. Tampering with the order of documents alters their novelty scores. This can potentially influence detection performance. We apply the LSH-FSD system by Petrovic et al. (2010) to the original version of Cross-Twitter 115k and to the time stamp corrected version. The FSD system achieves a 3% lower detection accuracy on the time stamp corrected version. This shows that re-ordering the data set impacts detection accuracy. We cannot detect an obvious pattern that would explain the rearrangement of document positions. Consequently, we cannot conclude why the original order results in lower detection accuracy than the time stamp corrected version.

We discovered that the document order in Cross-Twitter 115k has been rearranged and showed that the incorrect order positively impacts detection performance. Nonetheless, we chose to report our experiments on the original order without correcting any document positions. Our decision is based on two factors: 1) it ensures coherence and comparability with published FSD performance results; 2) we only use Cross-Twitter 115k to compare the performance of FSD systems with each other and do not make general statements on their accuracy.

Cross-Twitter 52 mio

We use the full version of Cross-Twitter, containing 52 million tweets, to conduct scaling experiments and investigate the behaviour of algorithms when applied to large amounts of documents. Note that Cross-Twitter is several orders of magnitude larger than the original TDT streaming data sets.

3.3 Baselines

The following section describes the two main baselines used in our performance evaluations of FSD systems. In particular, we make use of the UMass system by Allan et al. (2002) and LSH-FSD by Petrovic et al. (2010).

UMass

UMass (Allan et al., 2002) is a state-of-the-art FSD system, known for its high detection accuracy in the proceedings of the TDT competitions (Petrovic, 2012). Because of its high accuracy, UMass system is commonly used as an FSD baseline (Petrovic, 2012; Wurzer et al.; 2015 Moran et al. 2016). Algorithm 3 outlines the exact pseudo code of the UMass FSD system, whose source code has been made available to us.

UMass relies on comparison based novelty computations. This means that a document's novelty is based on 1 minus the maximum similarity to all previously encountered documents. UMass exhaustively determines maximum similarity by comparing a document arriving from the stream with all previously arrived documents using cosine similarity. To speed up processing time, an inverted index is used, which allows comparing new documents to only those documents that share at least one term in common, as seen at Line 3 in Algorithm 3. The number of terms that both documents share in common determines the number of scalar multiplications. Once the most similar previous document is identified, UMass emits the inverse of its cosine similarity as the novelty score.

Algorithm 3 : UMass FSD System

```
1: for all $document\ d \in Stream$ do
2:     for all $term\ t \in d$ do
3:         for all $document\ d' : t \in d'$ do
4:             update cosine similarity(d,$d'$)
5:         end
6:     end
7:     $sim_{max} \leftarrow max_{d'}\{\text{cosine similarity}(d,d')\}$
8:     noveltyScore(d) $\leftarrow (1 - sim_{max})$
9: end
```

The UMass system does not scale because new documents are compared with all previously encountered documents to determine the closest one. Consequently, the time and space complexity grows linearly ($O(n)$) with respect to the number of documents processed. We observe the impact of continuously growing numbers of documents on UMass's throughput in Chapter 7. The inverted index reduces the average time complexity per document from $A(n*dl)$ (exhaustive comparison to all previous documents) to $A(n*dl/v)$ (comparison with only those documents that share common terms) operations. Whereas dl refers to the average document length, n to the number of documents processed in the past and v to the number of unique terms found in the n previous documents. While studying the source code of UMass, we discovered that recent document are prioritized over older documents when computing novelty. This is not mentioned in any of UMass publications. Our experiments in Chapter 6 reveal that

UMass inflates the similarity scores of recent documents, which results in increased detection accuracy. By default the UMass system operates on hard drive. To make our experiments comparable to other systems, we always adjust the default setting to operate in memory instead.

Although inefficient, comparing new documents exhaustively to all previously arrived documents, results in state-of-the-art detection accuracy. In this thesis, we make use of the UMass FSD system as a high accuracy baseline in our effectiveness experiments.

LSH-FSD

LSH-FSD by Petrovic et al. (2010) is currently the fastest FSD system that achieves accuracy comparable to exhaustive document comparisons (UMass). On a single core LSH-FSD is able to process around 500 tweets per second, which resembles 10% of the full Twitter stream. As UMass, LSH-FSD relies on comparison based novelty computation, where novelty scores are determined by 1 minus the similarity to the closest previously encountered document. Algorithm 4 shows the exact pseudo-code of LSH-FSD, whose source code has been made available to us.

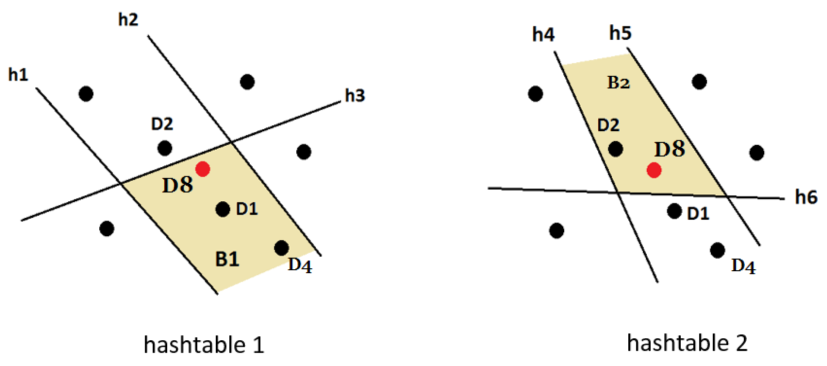


Figure 3.3: Comparing the observed document position in Cross-Twitter 115k with their true position, determined by their publication time stamp.

LSH-FSD achieves state-of-the-art efficiency in terms of throughput because it applies Locality Sensitive Hashing (LSH) to limit the search field when identifying the most similar previously encountered document. LSH is a randomized algorithm for approximated nearest neighbour search (Charikar, 2002). In FSD the nearest neighbour is the

Algorithm 4 : LSH-FSD System

```
1:  for all hashtables $T_m \in T : \{T_1 ... T_L\}$ do
2:      $T_m \leftarrow$ populate hyperplanes $\{H_1 ... H_k\}$
3:  end
4:  for all document $d_n \in Stream$ do
5:      candidates $\leftarrow 0$
6:      for all hashtables $T_m \in T$ do
7:          Bucket $B_{T,m} \leftarrow$ hash$(d_n, T_m)$
8:          candidates $\leftarrow candidates \bigcup B_{T,m}$
9:      end
10:     for all term $t \in d_n$ do
11:         for all $d' \in \{d' | t \in d', d' \in candidates\}$ do
12:             update cosine similarity$(d_n, d')$
13:         end
14:     end
15:     $sim_{max} \leftarrow max_{d'}\{$cosine similarity$(d_n, d')\}$
16:     if $sim_{max} < \alpha$ then
17:         for all term $t \in d_n$ do
18:             for all $d' \in \{d_{n-k} ... d_{n-1}\}$ do
19:                 update cosine similarity$(d_n, d')$
20:             end
21:         end
22:         $simVar_{max} \leftarrow max_{d'}\{$cosine similarity$(d_n, d')\}$
23:     end
24:     if $simVar_{max} > sim_{max}$ then
25:         noveltyScore$(d_n) \leftarrow (1 - simVar_{max})$
26:     else
27:         noveltyScore$(d_n) \leftarrow (1 - sim_{max})$
28:     end
29: end
```

most similar previously processed document, which determines the degree of novelty for a newly arriving document. Instead of comparing new documents to all previous ones, LSH generates candidate sets of documents that are likely being nearest neighbours (Indyk et al., 1998). More precisely, LSH randomly intersects the vector space using hyper planes. Initially, a fixed set of planes is populated by sampling from a zero centred Gaussian distribution. Figure 3.3 illustrates a simplified 2-dimensional view of a vector space with LSH using 3 hyper planes ($h1 - h3$) and 8 documents ($D1 - D8$). The resulting planes ($h1 - h3$) randomly intersect the vector space, forming hyper-polygonal shaped buckets, like bucket $B1$ - covering the colourized area. Documents are projecting onto all random hyperplane, which places them into their corresponding buckets. This process is referred to as hashing because it assigns every document the unique ID (hash code) of its bucket. The error bounds of LSH guarantee that documents that fall into the same bucket are more likely to be similar than documents in other buckets. Since LSH positions its hyperplanes randomly, it cannot guarantee to detect the true nearest neighbour. In Figure 3.3 document $D8$ is a new document arriving from the stream. LSH places document $D8$ in bucket $B1$ which identifies documents $D1$ and $D4$ as nearest neighbour candidates. Figure 3.3 reveals that the true nearest neighbour for document $D8$ is $D2$, which is missed. To increase the chance of capturing the true nearest neighbour LSH-FSD produces multiple hashtables, each with different positioned hyper planes. Figure 3.3 illustrates 2 hashtables. Note that the position of documents ($D1 - D8$) remain the same but the positions of the hyper planes ($h1 - h3$) in hashtable 1 and ($h4 - h6$) in hashtable 2 change. Documents are hashed using all hashtables, placing them into multiple buckets (one per hashtable). In hashtable 2 document $D8$ falls into bucket $B2$, which identifies document $D2$ as a nearest neighbour candidate. The final candidate set consists of the union of all target buckets ($B1, B2$) and identifies documents $D1$, $D2$ and $D4$ as nearest neighbour candidates for document $D8$. We set-up LSH-FSD to use 70 hash tables and 13 bit hash codes because these settings where found to be optimal by Petrovic when applied to the Cross-Twitter data set.

LSH-FSD applies a version of LSH that guarantees error bounds based on the cosine similarity. LSH is used to determined a new document's degree of novelty with respect to the past. Each document d_n arriving from the stream is projected (hashed) onto the hyperplanes in all hashtables, placing it into a certain bucket. These buckets already contain previous documents, which are likely to be similar to d_n and form the candidate set (Algorithm 4, Lines 4 - 8). LSH-FSD exhaustively compares all documents in the candidate set (union of all target buckets) to the new document to identify the most similar one - the true nearest neighbour (Algorithm 4, Lines 9 - 13). Although LSH-FSD uses multiple hashtables, LSH by itself often fails to identify the true nearest neighbour. To compensate for the poor performance of LSH, LSH-FSD additionally checks the most recently encountered *k* documents, whenever LSH does not yield a sufficiently similar document (Algorithm 4, Lines 15 - 22). If exhaustive comparison with the *k* most recent document reveals a document more similar to the new document than the nearest neighbour according to LSH, LSH-FSD discards the LSH candidate and computes novelty based on the closest found document (Algorithm 4, Lines 23 - 27).

LSH by itself only reduces the search space but does not guarantee constant performance as each new document needs to be stored in all its corresponding buckets in all hashtables for future comparisons. To guarantee scalability, LSH-FSD places an upper limit on the number of documents per bucket and removes the oldest document whenever a predefined limit is reached. This allows triggering a deletion procedure, whenever a particular region in the vector space becomes too densely populated. This deletion strategy is also referred to as “local deletion” and found to be superior to temporal deletion for FSD on Twitter (Petrovic, 2013). According to Petrovic, limiting the number of documents in all hashtables results in constant (O(1)) time and space complexity with respect to the number of documents processed. We however show in Chapter 5 that LSH-FSD grows slightly but continuously in space. The average number of operations per document is $A((|B_{avg}|+k)*t)$, where $|B_{avg}|$ denotes the average number of documents per bucket, k denotes the number of additionally checked documents and t denotes the aver document length.

The combination of search space reduction through LSH with local deletion drastically increases the detection throughput of FSD systems without sacrificing effectiveness. LSH-FSD achieves state-of-the-art FSD performance in effectiveness and efficiency (throughput and lag). In this thesis, we use LSH-FSD as our main baseline and compare our approach to it for high effectiveness and efficiency and scaling experiments.

3.4 Hardware and Parameter Setting

Unless stated differently, all our experimental results are reported on an idle machine using a single Intel-Xeon CPU core with 2.27GHz. All reported runtimes are averaged over 5 runs to additionally reduce possible perturbation from the OS.

We always report the accuracy of LSH-FSD with optimal parameter setting. LSH-FSD performs best on Cross-Twitter when using 70 hash-tables and 13 bit bash codes. Note these was also the default parameter setting reported by Petrovic (2012). UMass does not offer any parameters for tuning detection accuracy.

3.5 Conclusion

This chapter outlined the data set, Cross-Twitter, which we use for our FSD experiments. Cross-Twitter provides a small (115k) and large (52 mio) version. Both versions share the same topics and only set themselves apart by the number of unlabelled data. We make use of Cross-Twitter-115k for effectiveness experiments, which allows algorithms to finish within an hour. Cross-Twitter-52mio results in processing times $>$ 1 day and is only used for scaling experiments. The accuracy between both data sets is comparable.

Furthermore, this chapter introduced the two strong main baselines and outlined their pseudo-code in detail. UMass, which is known for its high effectiveness, is our benchmark for detection accuracy. LSH-FSD, the currently fastest FSD system, scales to millions of documents while maintaining state-of-the-art effectiveness and efficiency, is our benchmark for detection efficiency.

Chapter 4

Memory-based Novelty Detection

4.1 Motivation

The only published method family that accurately detects novelty without lag, is comparison based. These methods provide state-of-the-art FSD effectiveness and efficiency, making them the default method for novelty computation. Unfortunately, the throughput of all published FSD systems falls short of the volume of modern data streams. The main challenge that must be tackled is to provide high scalability and throughput without sacrificing accuracy or increasing lag.

Efficiency

By March 2016, Twitter publishes on average 5,787 tweets every second[1]. Even the currently fastest published FSD system, LSH-FSD by Petrovic et al. (2010), can only cope with less than 10% of the average Twitter volume when running on a single core. This is problematic, as outlined in Chapter 1, because the value of detected information depends on the lag it can be detected with. McCreadie et al. (2013) scaled LSH-FSD to reach a level of throughput comparable to that of the average Twitter volume. In particular, they scaled LSH-FSD by parallelization using a Storm[2] cluster of 72 cores (unknown CPU model). The high number of cores is necessary to distribute the workflows of LSH-FSD over a cluster.

[1]https://about.twitter.com/company (last updated: March 31, 2016)

[2]Apache Storm http://storm.apache.org/ (last updated August 31, 2016)

In order to apply FSD systems on modern sized data streams, novelty detection must be able to process thousands of documents per second without sacrificing accuracy or introducing lag.

Scalability
FSD is by definition a streaming task. Unbounded data streams demand constant operation in terms of time and space with respect to the number of documents processed. This requires FSD systems to compute novelty with respect to an ever growing set of documents at a constant cost per-document. State-of-the-art novelty detection methods fail to provide true scalability because they either come at a significant cost in accuracy or can't guarantee constant performance.

In this chapter, we introduce a new algorithm family for novelty computation called memory-based methods. Note that in this thesis, memory-based methods do not address the temporal storage unit in a PC, but the faculty by which the mind stores and remembers information. memory-based methods are non-comparison based - they do not compare a document with previously seen documents individually to compute novelty. Instead, a single representation of the past - the memory - is constructed, capturing the information of all encountered documents. This allows determining a document's novelty with respect to a single representation instead of individual documents. Reducing the past to a single representation enables memory-based algorithms, to vastly outperform comparison based novelty computation in terms of efficiency without sacrificing effectiveness.

Instead of scaling existing non-constant algorithms $(O(n))$ to unbounded data streams, this thesis introduces an entirely new and highly efficient algorithm family, specifically targeted at streaming environments that achieves constant operation of $(O(1))$ in terms of time and space, by design.

4.2 Intuition on memory-based Novelty Detection

memory-based novelty detection mimics the procedure of novelty detection as it is carried out by humans. Imagine a person is presented the task of novelty detection. He reads tweets one at a time and determines, whether they speak about previously unseen events. When this person is confronted with a new document, like the tweet in Figure 4.1, he will spot and extract "*concepts*".

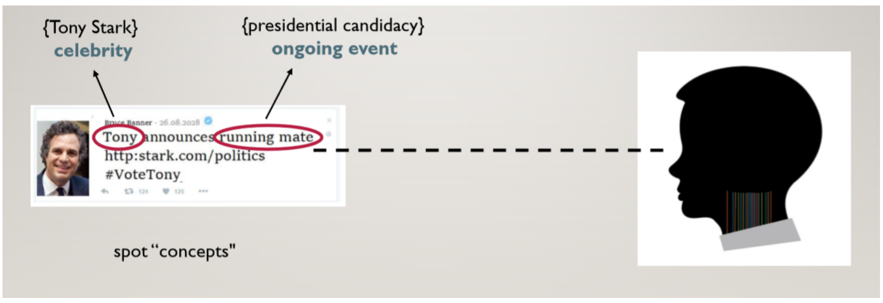


Figure 4.1: Illustration of the human novelty detection process: Whenever a new document is presented, concepts are spotted.

In the case of the example tweet, the concepts appear in the form of a familiar name, which refers to a celebrity and the term "running-mate", which relates to the concept of an ongoing election, during which running-mates are announced. Once the person spots a concept, he relates it to those in his memory to determine whether the presented information is familiar. This process is illustrated by Figure 4.2. Whenever a concept is unfamiliar i.e. it does not appear in his memory, the person will recognize it as new information. Previously unfamiliar concepts are subsequently remembered when they can't be related to any of the familiar concepts of the memory.

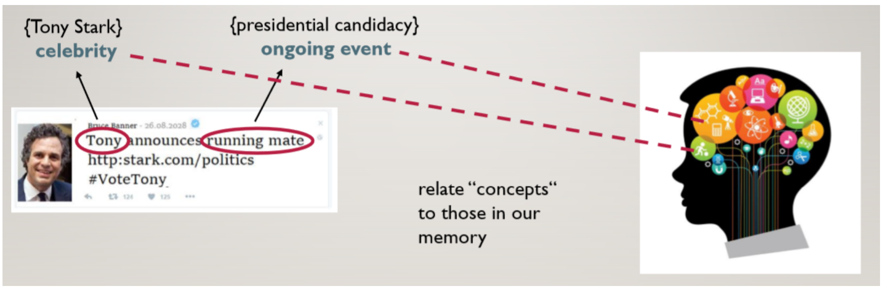


Figure 4.2: Illustration of the human novelty detection process: The Spotted concepts are related to those in the memory.

The human novelty detection process incorporates three major steps:
1) spotting the concepts
2) relating the spotted concepts to those in the memory
3) remembering previously unfamiliar concepts in the memory

Our new algorithm family called “memory-based Novelty Detection” mimics these three steps of the human novelty detection process.

In this chapter, we develop kterm hashing, the first member of the memory-based novelty detection family. Kterm hashing describes a distinct methodology on how to solve the three steps of memory-based novelty detection. Depending on the application, there are various approaches viable for each step. For the remainder of this chapter we focus on novelty detection through kterm hashing for the purpose of First Story Detection. Note that the principle of memory-based novelty detection and kterm hashing are applicable beyond FSD, as shown in Chapter 8, where we apply kterm hashing to evaluate sentence entailment.

The main challenge of applying memory-based novelty detection to FSD arises from the strong constraints of the streaming environment that come with the research task. These include single pass processing in constant space and time while delivering a throughput comparable to those of the full Twitter stream. Consequently, FSD requires memory-based novelty detection to perform each the three steps 5,000 times per second.

4.3 Kterm Hashing

We set out to construct a new algorithm family to compute novelty, called memory-based novelty detection. The remainder of this chapter is dedicated to construct kterm hashing, the first member of the memory-based novelty detection algorithm family. Kterm hashing overcomes the shortcomings of comparison based methods while retaining accuracy and zero lag. Consequently, kerm hashing requires the same amount of time and space to compute the novelty of the $1,000,000^{th}$ document as the 1^{th} document. This provides superior algorithm complexity over state-of-the-art comparison-based novelty detection algorithms, which require *n - 1* comparisons.

4.3.1 Constructing a Novelty Computation Method

We define the content c_n to be the set of all concepts T that can be found in the document $d_n : c_n = \{ T : T \in d_n \}$. We consider the content c_n to be the representation of all information in document d_n.

Kterm hashing, represents all past information in its memory. More formally, the memory M consists of the contents $(c_1...c_{n-1})$ of all documents $(d_1...d_{n-1})$ encountered before document d_n and is denoted by M_{n-1}.

For each document d_n arriving from the stream, we construct its content c_n based on the concepts in d_n and compute its novelty with respect to the memory of the past (M_{n-1}). After the degree of novelty is computed, we update the memory M to include the content c_n using Equation 4.1.

$$M_n \leftarrow M_{n-1} \cup c_n \tag{4.1}$$

Hypothesis 1: *The number of unseen terms in a document is a better indication of its novelty than random decision making*

Hypothesis 1 assumes that each term represents a concept and the more new concepts/terms a document contains with respect to all previously seen documents, the more novel it is. This represents the key idea of kterm hashing, which computes the novelty of document d_n based on the novelty of its content c_n with respect to memory M_{n-1}, which holds previous information.

We test Hypothesis 1 by building an FSD system and applying it to Cross-Twitter 115k. We estimate the novelty of each document d_n in the data stream based on Equation 4.2. This equation calculates the novelty of document d_n and its content c_n based on the number of unseen concepts/terms T with respect to the memory M_{n-1}.

$$Novelty(c_n) = \sum_{T \in c_n} \begin{Bmatrix} 1 : T \notin M_{n-1} \\ 0 : T \in M_{n-1} \end{Bmatrix} \tag{4.2}$$

This leaves us with a novelty score for each document. To determine the minimum detection cost, we sweep the detection threshold, calculating false alarm and miss probabilities at each step. The optimal threshold is found when the detection cost is minimized. All documents, whose novelty score exceeds the detection threshold are considered to speak about a new event. All other documents are considered as follow-ups.

The normalised minimum detection cost of an FSD system based on Hypothesis 1 results in $C_{min} = 0.9752$. To set this in perspective, we compare it to a system with random performance. Based on a biased coin the random system either considers all documents to speak about new events or to be follow-ups. Random performance results in $C_{min} = 1$. As explained in Chapter 2, C_{min} is a cost metric, which means that lower values indicate better performance. Our experiment confirms Hypothesis 1, since determining a document's novelty based on the number of unseen concepts/terms outperforms random decision making. Note that due to the low number of topics in the Cross-Twitter 115k data set small differences of 3% are not significant.

Observation 1: *The longer a message is the more likely it is to contain new terms*

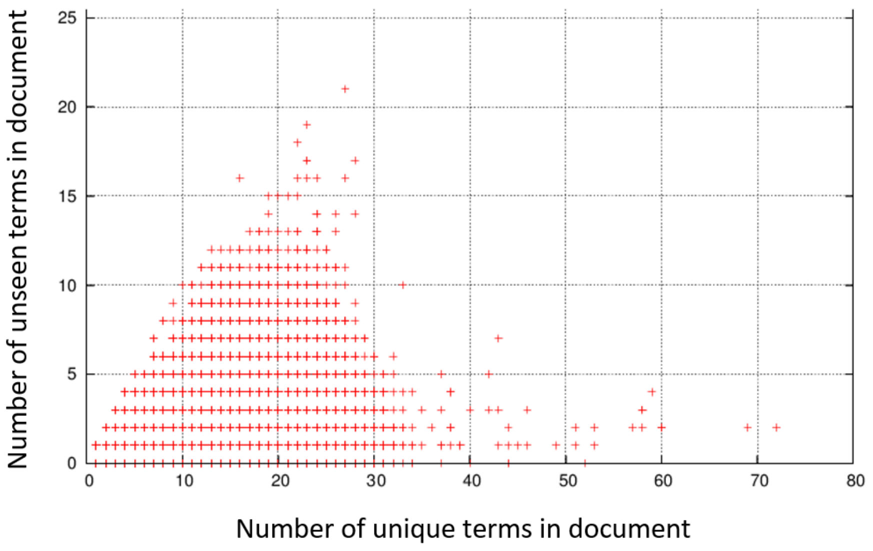


Figure 4.3: Illustration of the correlation between document length and novelty i.e. the number of unseen terms with respect to the past.

When computing novelty using Equation 4.2, we observed a correlation between a document's length and novelty, as seen in Figure 4.3. Documents on social media vary in length and range from a single term to dozens of terms.

Refined Hypothesis 1: *The number of unseen terms in a document in proportion to its length, is a better indication of its novelty than random decision making*

To test refined Hypothesis 1 and avoid longer documents from overpowering shorter ones, we normalize a content's novelty score by its size. In particular, we compute the fraction between the novelty of the content c_n and its size $|c_n|$, as seen in Equation 4.3.

$$Novelty(d_n) = \frac{Novelty(c_n)}{|c_n|} \tag{4.3}$$

We test the refined Hypothesis 1 by computing the novelty score of Cross-Twitter 115k. The document novelty now corresponds to the size-normalized content novelty, as seen in equation 4.3. The size of content c_n is denoted by $|c_n|$ and is the number of unique terms found in document d_n. The minimum detection cost improves by 1.2% to $C_{min} = 0.9633$, which confirms the refined Hypothesis 1.

Hypothesis 2: *Unseen combinations of terms provide a better indication of a document's novelty than unseen terms in isolation.*

Note that up to this point we modelled concepts by single terms, which implies a full independence assumption. Hypothesis 2 represents a powerful core idea of kterm hashing by stating that even if individual words appeared in a document before, encountering them in a new combination, also indicates novelty. For example, encountering the terms "fire" and "Louvre" individual is not unlikely, as both are concepts that humans might talk about. However, encountering them together in a single message might indicate an outbreak of a new event. Instead of computing novelty based on the presence of unseen terms, we assume a sequential dependence assumption (Metzler & Croft, 2005) and form n-grams up to length 3 to test Hypothesis 2. N-grams are created by combining adjacent terms (sequential dependence), as seen in Table 4.1.

We compute the novelty of all documents in Cross-Twitter based on Equation 4.3. However, this time we define the content c_n by n-grams up to length 3 instead of individual terms. We confirm Hypothesis 2 since measuring a document's novelty based on combinations of terms, decreases the normalised minimum detection cost by 5.7%

document	some are more equal than others
1-gram	*some, are, more, equal, than, others*
2-gram	*some-are, are-more, more-equal, equal-than, than-others*
3-gram	*some-are-more, are-more-equal, more-equal-than, equal-than-others*

Table 4.1: Example sentence with corresponding n-grams of length 1 to 3

($C_{min} = 0.9081$) in comparison to the refined Hypothesis 1. Note that this result confirms the findings of Metzler & Croft (2005) and Hasan et al.,2008. Metzler & Croft (2005) found sequential term dependence superior to full term independence for IR retrieval tasks and Hasan et al. (2008) improved translation quality of statistical machine translation by modelling term "triplets".

Hypothesis 3: *A full term dependence assumption provides a better model for novelty detection than a sequential term dependence assumption.*

So far, we modelled concepts by n-grams that assume sequential term dependencies. This preserves the order in which the terms appeared in the original document. Metzler & Croft (2005) found a full term dependence assumption beneficial for retrieval tasks on less homogeneous collections and short queries. These circumstances are closely related to the once we find when computing novelty detection on stream. We therefore hypothesize that a full term dependence assumption also positively influences the accuracy of kterm hashing when detecting novelty.

To test Hypothesis 3, we shift away from n-grams and break the sequential dependence assumption. We coin the word "*kterm*", which we define to be a compounded term resulting from forming all possible combinations of all terms found in a document. More formally, we denote a kterm $T = \{t_1, t_2, ...\}$ to be a non-empty set of up to k distinct terms t. We distinguish between different lengths of kterms, which we denote by $|T|$. Table 4.2 shows an example document and its corresponding kterms up to $k = 3$.

Kterms distinguish themselves from n-grams by ignoring the order in which terms appear in a document, as seen in Table 4.2. Instead of grouping adjacent terms, kterms form sets of all possible combinations.

document	some are more equal than others
1-terms	$\{are\};\{equal\};\{more\};\{others\};\{some\};\{than\};$
2-terms	$\{are,equal\};\{are,more\};\{are,others\};\{are,some\};\{are,than\};$ $\{equal,more\};\{equal,others\};\{equal,some\};\{equal,than\};$ $\{more,others\};\{more,some\};\{more,than\};\{others,some\};$ $\{others,than\};\{some,than\};$
3-terms	$\{are,equal,more\};\{are,equal,others\};\{are,equal,some\};$ $\{are,equal,than\};\{are,more,others\};\{are,more,some\};$ $\{are,more,than\};\{are,others,some\};\{are,others,than\};$ $\{are,some,than\};\{equal,more,others\};\{equal,more,some\};$ $\{equal,more,than\};\{equal,others,some\};\{equal,others,than\};$ $\{equal,some,than\};\{more,others,some\};\{more,others,than\};$ $\{more,some,than\};\{others,some,than\};$

Table 4.2: Example sentence with corresponding kterms of length 1 to 3

We still compute a document's novelty by the size-normalized content novelty seen in Equation 4.3. Instead of single terms or n-grams we now form kterms.

The size of content c_n, denoted by $|c_n|$, depends on the number of kterms that can be formed from the terms in document d_n. To compute the number of kterms, we distinguish them by their size, denoted by $|T|$ - the number of compounded terms. For a particular document length $|d_n|$ and kterm size $|T|$, the number of possible kterms equals to $\binom{|d_n|}{|T|}$. To calculate the size of the entire content c_n we sum up the number of possible kterms per size, as seen in Equation 4.4.

$$|c_n| = \sum_{|T|=1}^{k} \binom{|d_n|}{|T|} \tag{4.4}$$

The final Equation 4.5 for computing the novelty of document d_n according to Hypothesis 3 is formed by combining Equation 4.3, 4.2 and 4.4.

$$Novelty(d_n) = \sum_{T \in c_n} \binom{|d_n|}{|T|}^{-1} \begin{Bmatrix} 1 : T \notin M_{n-1} \\ 0 : T \in M_{n-1} \end{Bmatrix} \tag{4.5}$$

When computing the novelty for all documents in Cross-Twitter 115k using kterms based on Equation 4.5, we measure a normalised minimum detection cost of $C_{min} = 0.835$. We confirm Hypothesis 3, since it improves the detection accuracy by 8% in comparison with Hypothesis 2. Also note that a normalised minimum detection cost of $C_{min} = 0.835$ is significantly better $(p < 0.05)$ than those of a random FSD system and statistically indistinguishable from UMass, a state-of-the-art and highly effective FSD system.

We name novelty computation resulting from Hypothesis 1 to 3 "*kterm hashing*". Kterm hashing computes a document's novelty with based on the fraction of unseen kterms to its length. This allows computing novelty with respect to a set of other documents via a single point of comparison - the memory. The memory captures all preserves the information of the past by storing all previously encountered kterms. A single point of comparison overcomes the need for individual document based comparisons that limits the efficiency of comparison based novelty detection methods.

4.3.2 Kterm Cardinality

Kterm hashing distinguishes kterms based on their length. We denote a kterm's length by its cardinality, which is determined by the number of words compounded when forming it. Remember that the novelty of a document's content is based on the number of unseen kterms with respect to the past. Kterms of high cardinality capture large portions of documents, which are less likely to appear in the memory than kterms of lower cardinality. Consequently, the absence of high cardinality kterm should contribute more to a document's novelty score. We expand Equation 4.5 by a weight parameter α to address the importance of kterms of different cardinalities, as seen in Equation 4.6.

$$\mathrm{N}(d_n) = \sum_{T \in c_n} \alpha_{|T|} \binom{|d_n|}{|T|}^{-1} \begin{Bmatrix} 1 : T \notin M_{n-1} \\ 0 : T \in M_{n-1} \end{Bmatrix} \tag{4.6}$$

Because the number of possible variable combination is high, we optimize the parameters $\alpha_1 ... \alpha_k$ on a training set using grid search. Our experiments show that the use of kterms of higher cardinality positively influences novelty detection. Increasing the kterm cardinality also increases the number of kterms formed. The content size is determined by the binomial coefficient between the length of a document and its cardinality, as seen in Table 4.4. Increasing the kterm cardinality sharply increases the content size, which reduces efficiency and increases the size of the memory. Luckily, the increase is capped by the document length. On tweets, whose average length is 10, the maximum number of kterms is formed when the cardinality is 5. Table 4.3 shows the memory size, which represents the number of kterms formed when processing 100,000 tweets. The table outlines why high cardinalities are suboptimal for high

kterm cardinality	memory size
1	$1 * 10^6$
1-2	$5.5 * 10^6$
1-3	$1.75 * 10^7$
1-4	$3.85 * 10^7$
1-5	$6.37 * 10^7$
1-6	$8.47 * 10^7$

Table 4.3: Table shows the size of the memory for different cardinalities of kterms when processing 100,000 tweets

efficiency applications.

name	**number of terms**
original tweet	10
1-kterm	10
2-kterm	45
3-kterm	120
4-kterm	210
5-kterm	252
6-kterm	210
7-kterm	120
8-kterm	45
9-kterm	10
10-kterm	1

Table 4.4: Table illustrating the content size for different kterms cardinalities; originating from an average length tweet of 10 distinct terms

4.3.3 Implementing Kterm hashing for novelty detection

Algorithm 5 outlines the detailed pseudo-code for novelty detection using kterm hashing. High-volume data streams require operation in low latency. In novelty detection, we listen to a stream of documents (*Algorithm 5, line 2*) with the intent to compute a novelty score for each document with respect to all previous documents.

For each document d_n arriving from the stream, we form its content c_n by creating all possible kterms up to a previously defined level of k (*Algorithm 5, line 3 & 4*). The novelty score $novelty_{d,n}$ for document d_n is computed based on the kterms in its content c_n using Equation 4.6, as seen in *Algorithm 5, line 5 - 12*. This requires a large number of membership checks with respect to the memory of the past M_{n-1}, one for each kterm in the content. Table 4.3 illustrates that even a short document in combination with a low kterm level k, spawns a large content due to its dependence on the binomial coefficient.

To keep operation constant with respect to space, we represent the memory M_{n-1}, preserving the contents of past documents $(d_1...d_{n-1})$, by a single bit-array. Each cell in the array holds a boolean value depending on whether the associated kterm was encountered before. We initialize the bit array with FALSE (0) values before the stream is processed (*Algorithm 5, line 1*). For the remainder of this thesis, we set the size of memory M to 300 megabytes, which allows storing a maximum of 2,516,582,400 entries. Using 300 megabytes allows kterm hashing to perform on par with UMass, a highly efficient state-of-the-art FSD system.

We ensure highly efficient membership checking of a kterm in the memory $(T \in M)$ by hashing each k-term $T : T \in c$ onto the bit-array (M). In particular, we compute 32-bit Murmur2 (Appleby, 2008) hashcodes for kterms, which efficiently address a sufficiently large array (*Algorithm 5, line 6*). Possible alternatives include SipHash (Aumasson et al., 2012) and CityHash (Pike et al., 2011). If the cell at the corresponding index is set to FALSE (0), the kterm is considered as novel (*Algorithm 5, line 7*). After checking the membership of a kterm in the memory, we flip the bit whenever the kterm was considered as novel (*Algorithm 5, line 9*). This results in membership check and memory update in a single step, which is possible because all kterms in a document's content are unique.

Algorithm 5 : memory-based FSD using kterm hashing

```
1: initialize memory $M$
2: for all $document\ d_n \in Stream$ do
3:     $t_{d_n} \leftarrow \{term\ t : t \in d_n\}$
4:     $c_n \leftarrow \{kterm\ kt : kt \subset t_{d_n}, |kt| \leq k\}$
5:     $M_n \leftarrow M_{n-1}$
6:     for all $kterm\ kt \in c_n$ do
7:         $kt_{hash} \leftarrow murmur(kt)$
8:         if $kt_{hash} \notin M_n$ then
9:             $novelty_{c,n} \leftarrow novelty_{c,n} + \alpha_{|kt|}$
10:            $M_n \leftarrow M_n \cup \{kt_{hash}\}$
11:        end
12:    end
13:    $novelty_{d,n} = novelty_{c,n} * \sum_{i=1}^{k} \binom{|t_{d_n}|}{i}^{-1}$
14: end
```

4.3.4 Memory Representation

The way we implement the memory mimics a primitive Bloom filter (Bloom, 1970) with only a single hash function. A potential downside of membership checking through hashing onto a bit array is that it introduces a small probability of false matches: a novel k-term kt_i may collide with a previously observed k-term kt_j and would be reported as non-novel. The probability of collision is directly proportional to the load factor of the bit array, i.e. the fraction of non-zero bits.

The probability that a particular bit is set 1 after inserting a single item into the bit array equals to $\frac{1}{m}$, whereas m denotes the size of the bit array. Consequently, the probability that a particular bit not set to 1 after inserting an item is $1 - \frac{1}{m}$.

$$P_{\text{particular bit is 1}} = 1 - (1 - \frac{1}{m})^n \tag{4.7}$$

Equation 4.7 shows the probability of a false alarm, which is the probability that after inserting n items into a bit array of size m, a particular bit is still 0. Given a desired false positive rate and an estimated upper limit on the number of elements n added to the bit array, one can estimate its size m given Equation 4.8.

$$m = \frac{1}{(1 - P_{\text{particular bit is 1}})^{\frac{1}{n}}} \tag{4.8}$$

Note that the used data structure is suboptimal in terms of false positive rate in comparison with a traditional Bloom filter. This arises from the use of a single hash function, compared to the multiple hash functions of a Bloom filter. Equation 4.9 shows the false positive rate of a Bloom filter with multiple hash (Fan et al. 2000; Bose et al. 2008;).

$$\text{False positive rate(BF)} = 1 - (1 - \frac{1}{m})^{n*k} \tag{4.9}$$

Here, k denotes the number of hash functions used when determining the hash bin. Each additional hash function decreases the likelihood that two different items collide. Unfortunately, every additional hash function also increases the time constant in kterm hashing by the number of the content size. It is important to notice that although the time constant is increased, it still remains constant with respect to the number of documents processed and therefore full-fills the requirements of streaming environments.

When designing memory-based methods, we aimed on scaling novelty detection to the equivalent of the entire Twitter stream. Consequently, our implementation of kterm hashing for novelty detection trades false positive rates for speed and operates with a single hash function. Interestingly, kterm hashing proved to be robust with respect to the number of false positives, since we cannot measure a significant advantage when using a collision free data structure. We conjure that the high numbers of unseen and total kterms per document mitigate the absence of multiple hash functions. Remember, kterm hashing estimates a document's novelty by the ratio of unseen to all kterms.

For the remainder of this thesis, we set the size of memory M to 300 megabytes, which allows storing a maximum of 2,516,582,400 entries. Using a single hash function and a memory of 300 megabytes, we achieve a higher detection accuracy ($C_{min} = 0.835$) than LSH-FSD ($C_{min} = 0.9061$). We elaborate on effectiveness experiments in Chapter 5.

4.4 Algorithm Complexity

Unbounded data streams require us to make a constant number of operations when determining the novelty of document d_n. This demands the runtime and storage cost for novelty computation to be independent of the number of documents ($d_1...d_{n-1}$) seen in the past.

Constant Time
The time complexity of novelty detection depends on the number of operations required to calculate the novelty score of a document. memory-based novelty detection methods, like kterm hashing, estimate novelty based on membership checks on the memory. The number of operations needed to determine the novelty score corresponds to the number of membership checks, i.e. the size of the content. The average time complexity $A(avg(|c|))$ is determined by the average content size, which depends on the average document size $avg(|d|)$ and the k, the maximum level for kterms. Both values are independent of the number of documents processed before. Consequently, kterm hashing results in a worst case time complexity of $O(1)$ with respect to the number of documents processed, when the document length is limited. By contrast, the worst case time complexity of traditional FSD systems is $O(n)$, where throughput decreases linearly with the number of documents processed (n).

Constant Space
The number of documents arriving from data streams and with it the size of the memory, grows without bound. To remain feasible, we represent the memory by a fixed-length bit array. Constraining the memory to a fixed data structure enables kterm hashing to maintain constant space, irrespective of the size of the stream it operates on. The resulting space complexity of kterm hashing is O(1) with respect to the number of documents processed. By Heaps law (Egghe, 2007) the number of distinct words - and subsequently the number of k-terms, will continue to grow and eventually saturate the bit-array. To mitigate this problem, we introduce a deletion strategy: whenever the load factor exceeds a pre-determined threshold ϕ, we zero out a random bit in the memory M. This allows us to keep the probability of false matches low, at the cost of forgetting some previously-seen k-terms. We explore the impact of deletion on FSD performance in Chapter 5 in detail.

4.5 Conclusion

In this chapter, we introduced a new algorithm family for novelty detection, called memory-based novelty detection. We further introduced kterm hashing, a memory-based novelty detection method. The key idea of kterm hashing is to resemble the past by a single representation - the memory. Novelty is determined by the fraction of unseen kterms with respect to the memory. In contrast to state-of-the-art comparison based methods, kterm hashing operates in truly constant time and space with respect to the number of documents processed.

Chapter 5

Improving FSD through memory-based Novelty Detection

5.1 Motivation

The previous chapter introduced a new novelty detection method called kterm hashing, which is able to operate in constant time and space. In this chapter, we apply kterm hashing to FSD and explore the impact of pre-processing as well as the cardinality of kterms on FSD effectiveness. Additionally, we compare the FSD performance of kterm hashing in terms of effectiveness and efficiency with two state-of-the-art FSD detection systems. To improve readability we refer to an FSD system, whose novelty detection method is based on kterm hashing, as kterm.

5.2 Impact of Preprocessing on FSD Performance using Kterm Hashing

Before we compare kterm with other FSD systems we explore the impact of several pre-processing methods on its FSD performance. Table 5.1 summarized the impact of individual pre-processing steps on the detection performance. We measure performance using the normalized minimum detection cost (C_{min}), which we introduced in the evaluation section in Chapter 2. Note that this is a cost metric where lower values indicate better detection accuracy.

	raw tweets	no Punctuation	no Username	no Links	no Hashtags
C_{min}	0.9637	0.8830	**0.8643**	0.9683	0.8731

Table 5.1: Impact of individual pre-processing steps on FSD performance using kterm hashing

Punctuation

We remove any character that is not a letter or number including the Twitter-specific hashtag and user symbols by splitting term on them: "#super-set" $\rightarrow$ "super", "set". According to Table 5.1, removing punctuation from tweets decreases the detection cost by -8% (from 0.9637 to 0.8830). Although removing punctuation characters is a standard preprocessing step in many Information Retrieval applications, it is not obvious that the same applies to FSD using kterm hashing. Especially URLs, which are commonly used on Twitter to circumvent its 140 character limit, spawn a high number of new terms when splitting on punctuation characters:
i.e. "http://www.manetas.com/pollock/" $\rightarrow$ "http", "www", "manetas", "com", "pollock". We conclude that the observed positive impact results from reducing the vocabulary mismatch by removing commonly used punctuations i.e. "'()!,?.;:. Splitting URLs on punctuation creates "garbage" terms, which are likely unseen. Unseen terms artificially inflate novelty scores computed by kterm hashing. We hypothesize that splitting hyperlinks on punctuation negatively impacts FSD accuracy.

Hyperlinks

To confirm our hypothesis, we remove punctuation from all terms except URLs. This prevents splitting them into "garbage" terms. Interestingly, we observe a slightly higher detection cost ($C_{min} = 0.8875$) than when splitting URLs. We additionally remove URLs entirely, which increases the detection cost to ($C_{min} = 0.9683$), as seen in Table 5.1. Both observations do not confirm our hypothesis, since splitting URLs does not harm detection accuracy following artificially inflated novelty scores. Manual inspection of Cross-Twitter revealed that no first story (detection target) includes an URL, whereas URLs are commonly found in follow-ups. This appears reasonable as external resources, like reports and official articles, are only available with a certain delay. According to Petrovic et al. (2013b) and Osborne et al. (2014), Twiter does not consistently lead newswires for reporting breaking events. Consequently, we do not build features based on the absence of URLs. We further investigated the impact of splitting URLs on punctuation characters in the follow-ups. Contrary to our intuition, splitting URLs produces more seen (i.e. protocol and country identifier) than unseen terms (i.e. domain and sub-directory names). Consequently, splitting URLs artificially increases the number of previously seen terms. The binomial coefficient further amplifies the artificially deflated proportion of seen terms, which results in lower novelty scores through kterm hashing. In Cross-Twitter only follow-ups contain URLs. Splitting them deflates their document's novelty scores, which increases FSD detection accuracy.

Usernames

Twitter allows its users to link other users to their messages by adding their username with a leading "@" symbol. This is a popular feature on Twitter as it allows directing messages to specific users. Usernames are unique identifiers and subsequently introduce new terms, which inflate novelty scores using kterm hashing. We found that in our dataset Cross-Twitter no first story contains usernames, whereas they are commonly found in follow-ups. We conjecture that once a topic has gained traction it becomes more likely that people talk about it and share it with their peers using their usernames. Table 5.1 shows that the detection cost drops by 10% when usernames are removed. Note that this is the highest positive impact we observed compared to all other preprocessing steps. Manual inspection showed that occasionally users tend to add a high number of other users to their tweets, resulting in more than 60% unseen terms with respect to all previous documents in the stream. Those tweets receive high novelty scores causing kterm to mistake them for new events.

Hashtags

Hashtags are Twitter-specific terms that link tweets to topics. Hashtags make tweets globally searchable by other users and allow users to discuss a topic publicly. Every user can reuse existing or create his own hashtags by adding terms with a leading hash symbol "#". Hashtags often consist of compounded terms (like kterms) i.e. #lifeIs-Great.

Because hashtags indicate topicality they are a commonly used features when identify new events in social media streams. Cataldi et al. (2010), Phuvipadawat & Murata (2010) and Ozdikis et al. (2012) for example rely on hashtags for new event detection. We however find that removing all hashtags decreases the detection cost of kterm by

9%, as seen in Table 5.1. This result is surprising as it contradicts intuition and previous research findings, including those systems that rely on hashtags for new event detection. We manually inspected all annotated topics to find the position of new hashtags in each topic substream, as seen in Figure 5.1. Note that we refer to topics as TDT entities that include a first story and its follow-ups, which share the same topicality. Figure 5.1 shows that the majority $(> 60\%)$ of hashtags does not occur in the first story, but in the follow-ups and 6 out of 27 Cross-Twitter topics don't contain any hashtags. Those hash-tags are new with respect to all previously encountered documents in the full Cross-Twitter stream and consequently spawn many unseen kterms in the follow-ups. Unseen kterms inflate the novelty scores and explain why hashtags are harmful to FSD using kterm hashing. Petrovic (2012) tested the impact of removing hashtags on LSH-FSD on the same dataset, which resulted in increased detection cost. We conjecture that kterm hashing is more sensitive to unseen terms than the length normalization of cosine hashing used in LSH-FSD. We also removed the hash symbol i.e. "#election" $\rightarrow$ "election". We expected that this reduces vocabulary mismatch as hashtags might match normal terms, but our experiments showed no improvement in FSD detection accuracy.

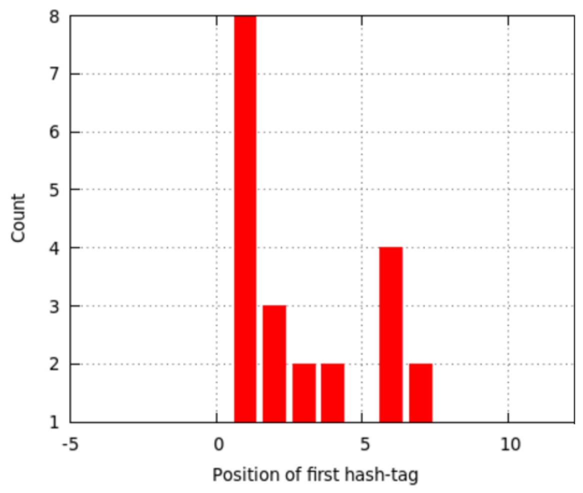


Figure 5.1: Illustration of the position of the first occurrences of hashtags within all annotated topics in Cross-Twitter. The majority of hashtags (13 out of 21) does not occur in the first story.

We further investigate the occurrence of new hashtags, their frequency and burstiness patterns. Figure 5.2 shows the document position of all new (previously unseen) hashtags within the topic substreams. Topic substreams cover only documents relevant to

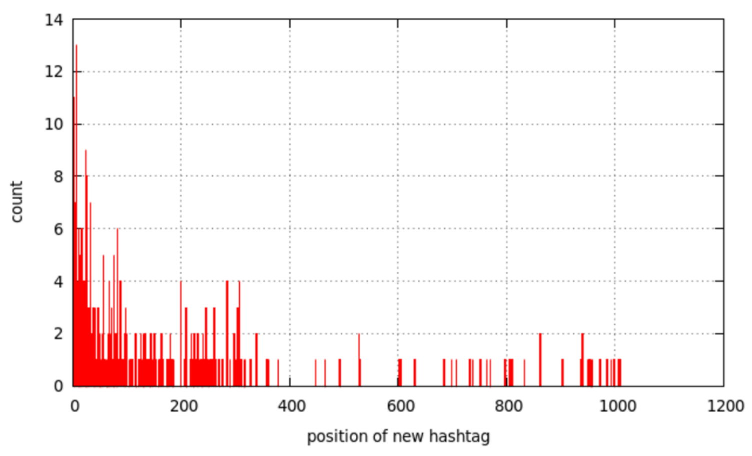


Figure 5.2: The x axis represents document positions within topics of Cross-Twitter. When topics are new, lots of new hashtags are introduced by people discussing them. The introduction of new hashtags flatten out later on.

the topics. For example, the first document of a topic X has position 1, the second document of the same topic has position 2. The graph shows that when the topic is new, users introduce a lot of new hashtags. Manual inspection showed that throughout the topic the usage of hashtags is roughly constant at about 10% of all tweets. We therefore conclude that Twitter users initially propose a lot of new hashtags before they settle on a few describing the topic best. To illustrate our conclusion, we retrospectively search for the most descriptive hashtag, according to human judgement for each topic. For example, the topic “Death of Amy Winehouse” is represented by “#RIPAmy”. This excludes unrelated or generally used hashtags like “#RIP” or “#whyher”.

Figure 5.3 plots the lag in hours on log-scale between the first story and the cumulative hashtag counts for all topics. Interestingly, we observe that none of the hashtags that occur in the first stories turn out to be the most descriptive one for their topics. They include hashtags like: #noooooo!, #breaking and #RIP or hashtags containing the only names of persons or companies involved. These hash-tags are not specific to any particular topic or event but appear to be features indicating the beginning of a new event, i.e. a first story. Approaches, like Phuvipadawat and Murata (2010), rely on similar hash-tags for event detection. They filter tweets tagged with the “#breakingnews” hashtag and perform event detection on them. Approaches that solely consider tweets with certain hashtags operate with zero lag and are highly scalable, following a heavily reduced stream volume. However, they are also limited in their accuracy, as they miss a large number of first stories. Although these hashtags appear to be useful features for FSD, we do not consider them in kterm hashing as their occurrences in our training topics is not sufficient to derive weights.

Figure 5.3 illustrates that average lag from a hashtag's first occurrence until it gets adopted by other users and experience frequency bursts, is 30 minutes. The graph also shows that some topics get adopted faster than others. The fastest rising hashtag in our dataset describes the death of a singer called Amy Winehouse. This topic also received the most attention among Twitter users, which we attribute to a general interest of Twitter users in news concerning celebrities. Financial news, like an acquisition by Google, reveal much slower and flatter growth rates. Although we only consider the most descriptive hashtags for each topic, all of them show very slow growth after their first mentions. This suggests that even popular hashtags grow initially slowly, before experiencing exponential growth. On average hashtags show burstiness patterns 1 hour after the corresponding first story occurs. This illustrates that detecting new events based on hashtags or their burstiness pattern is unsuitable for FSD because new events can only be detected with a substantial lag.

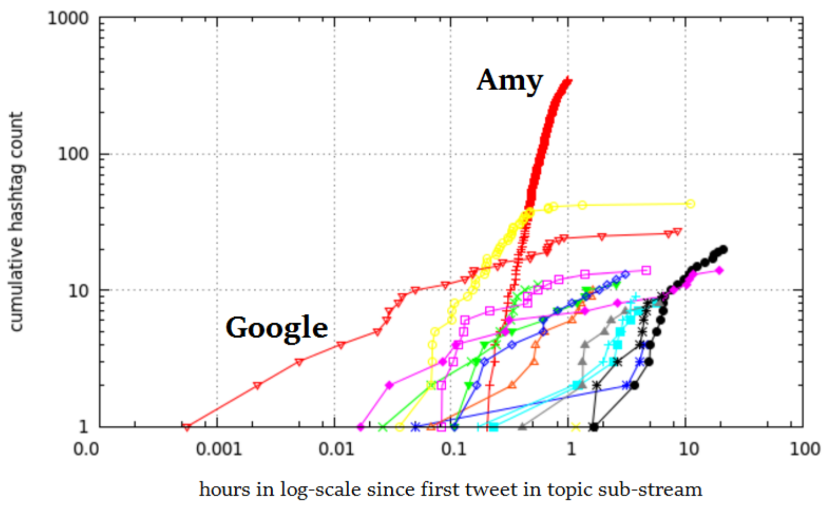


Figure 5.3: Illustration of the position of the first occurrence of hash-tags for all annotated topics in Cross-Twitter. The majority of hashtags does not occur in the first story.

For FSD using kterm hashing, we remove all hashtags because we found that they impact accuracy negatively on the Cross-Twitter dataset.

Stemming

Stemming reduces terms to their base-form i.e. “running” $\rightarrow$ “run”. Stemming is a commonly used preprocessing step in Information Retrieval applications to reduces vocabulary mismatch. We compare the impact of three different stemmers, Porter, Snowball and Krovetz on FSD effectiveness using kterm hashing. The resulting normalised minimum detection costs are visible in Table 5.2. We apply all stemmers after splitting terms on punctuation characters, which supports them to correctly identify the base forms. The table shows a positive impact on the detection cost for all three stemmers. The Porter and Snowball stemmer result in slightly higher performance (1% difference) than the Krovetz stemmer. Replacing terms by their base-form decreases vocabulary mismatching, which corrects the number of unseen terms and positively impacts the FSD cost using kterm hashing.

	punctuation removed	Porter	Snowball	Krovetz
C_{min}	0.8830	**0.8528**	**0.8528**	0.8630

Table 5.2: Impact of stemming on FSD performance by kterm hashing; Note that all stemmers are applied to tweets without punctuation.

Final Preprocessing

So far we explored impact of several preprocessing steps on the FSD cost using kterm hashing, in isolation. To decide on the final preprocessing procedure, we look at their impact when combining them. Table 5.3 shows the performance of kterm and several combinations of preprocessing steps. We observe that the positive effect of all tested preprocessing steps is additive, and combining them results in the lowest observed FSD cost of 0.8076. Our final preprocessing method consists of the removal of hashtags and usernames, splitting terms on punctuation and stemming using a Porter stemmer.

Pre-Processing	C_{min}
no Hashtags, no Username, plus URLs	0.8573
no Hashtags, no Username, plus URLs, no Punctuation	0.8324
no Hashtags, no Username, plus URLs, no Punctuation, plus Porter	0.8076

Table 5.3: Impact of combined pre-processing steps on FSD performance by kterm hashing

5.3 Impact of Cardinality on Kterm Hashing for FSD

Kterms are sets of compounded terms found in a document. The cardinality of a kterm is determined by the number of terms compounded when forming it, i.e. the size of the set. The kterm {“catch”, “them”, “all”} for example, has a cardinality of 3, because it is composed of 3 terms. The number of possible kterms depends on the document size and the maximum allowed cardinality. The cardinality of kterms is naturally limited by the size of the document, where a single kterm spans the entire document.

The maximum allowed cardinality of kterms determines the size of a document’s content - the set of all constructed kterms. Each kterm requires a lookup in the memory to determine whether it is unseen with respect to the past. Consequently, a kterm’s cardinality influences the efficiency of an FSD using kterm hashing. Additionally, we conjecture that the cardinality of kterms also influences their effectiveness.

5.3.1 Impact of Kterm Cardinality on Effectiveness

We explore the impact of varying the cardinality between 1 and 5 on FSD effectiveness, measured by the normalized minimum detection cost. Table 5.4 summarizes the performance of kterm on Cross-Twitter 115k for different levels of cardinality. Unsurprisingly kterms of cardinality 1 result in poor performance of $C_{min} = 0.9822$. This resembles simple counting what fraction of terms are new. Single terms cannot capture concepts of events, which usually combine multiple terms. Encountering the individual terms “fire” and “Louvre” is not unlikely as both are concepts that humans might talk about. However, encountering them together in a single tweet might indicate an outbreak of a new event. This explains why higher levels of cardinality result in lower detection costs. Kterms with Cardinality of 3 result in the lowest detection cost of $C_{min} = 0.8289$. Interestingly, when increasing the kterm cardinality beyond level 3, we measure the inverse effect, as detection cost increases. Manual inspection revealed that kterms of cardinality greater than 3 are rarer and often span entire documents. This observation partially confirms our hypothesis that kterm cardinality should influence the novelty score: up to a certain point (3 for tweets with an average length of 10 terms), kterms of higher cardinality are better suited for novelty detection.

Combining kterms of different cardinality

Up until now, we have looked at the performance of different level of cardinality in isolation and found that kterms compounded of 3 terms result in the lowest detection cost. Although other levels of cardinality performed worse, we assume that they are able to capture additional information that might be worth considering. In our next experiment, we stepwise increase the cardinality of kterms and combine them with all previously formed kterms. Table 5.4 summarizes the minimum detection cost C_{min} of cardinality 1-6. Note that combined C_{min} at a certain cardinality includes all lower levels as well i.e. combined C_{min} of cardinality 3, includes kterms of cardinality 1 to 3.

Cardinality	individual C_{min}	combined C_{min}	combined & weighted C_{min}
1	0.9822	0.9822	0.9822
2	0.8808	0.9306	0.8803
3	0.8289	0.9089	0.7902
4	0.8539	0.8958	0.7893
5	0.8549	0.8941	**0.7891**

Table 5.4: compares the detection cost (C_{min}) for different individual and cumulated cardinalities of kterms

Table 5.4 shows that the combination of different cardinalities undercuts their individual detection costs. This confirms our hypothesis that although certain levels of cardinality perform ineffective on an individual basis, they still capture valuable information when combining them. For now, we assume uniform weights for all levels of cardinalities. This means that kterms of all cardinalities contribute equally to the novelty score. The table shows that each additional level of cardinality further decreases the detection cost. Furthermore, we observe that the improvement of each additional step becomes progressively less. We assume that the average length of tweets (10) limits the impact of higher cardinalities.

Parametrising kterm cardinality

Our experiments showed that combining kterms of different cardinalities is beneficial for FSD effectiveness. So far we considered kterms of all cardinalities as equally important when combining them.

Extracting a previously seen kterm of high cardinality from a new document indicates that it shares several terms (those the kterm is composed of) with a previously encountered document. Furthermore, encountering an unseen kterm of high cardinality indicates that the current document carries a combination of several terms (the unseen kterm) that has not been seen in any of the previously encountered documents, which provides an indication of novelty. We conjecture that a kterm's contribution to the final novelty score should depend on its cardinality - the number of terms it is composed of.

When we formally introduce kterm hashing in Chapter 4, we provided Equation 5.1, which offers a dedicated weight (α) for kterms of different cardinality.

$$\mathrm{N}(d_n) = \sum_{T \in c_n} \alpha_{|T|} \binom{|d_n|}{|T|}^{-1} \begin{Bmatrix} 1 : T \notin M_{n-1} \\ 0 : T \in M_{n-1} \end{Bmatrix} \tag{5.1}$$

The weight allows distributing importance among the different levels of cardinality. We optimize weights for the normalized minimum detection cost C_{min}, defined in Equation 2.8 in Chapter 1, using grid search. All optimization is based on a training set of 10 topics, which we manually labelled. Those topic originate from the same list of important events[1] as the original Cross-Twitter topics and include natural disasters, political, financial and celebrity news. We optimize the parameters using grid search because the number of variables is low and their valid range is limited. The size of the training set, which consists of 10 positive and 2k negative training points, does not allow learning weights for individual kterm cardinalities using gradient methods. We show in Chapter 8 how weights for kterm hashing can be learned using an SVM, when sufficient training data is available. During grid search we alter the weights by

[1]https://en.wikipedia.org/wiki/2011

cardinality	optimal weight
1	-0.01
2	0.22
3	0.54
4	0.12
5	0.11

Table 5.5: optimal weight for each kterm cardinality according to grid search on our test set

0.01 and step between -1 and +1. The optimal weight setting is reached when C_{min} is lowest. The resulting optimal weight for cardinality of 1 to 5 are normalized to add up to 1 and can be found in Table 5.5.

5.3.2 Impact of Kterm Cardinality on Efficiency

In the beginning of this section we mentioned that each kterm requires a lookup in the Memory structure to determine whether is new with respect to all previously seen documents. Every lookup involves hashing the kterm onto a bit-array and a subsequent check of the corresponding entry. Combining several levels of cardinalities results in high effectiveness but also spawns a high number of kterms. Table 5.6 compares the trade-off between effectiveness and efficiency when combining different cardinalities. The table reveals that the marginal gains of cardinality > 3 come at a high cost in efficiency measured by the runtime in seconds. For example, when increasing the cardinality from 3 to 4, we decrease the minimum detection cost by 1% at the cost of multiplying the runtime by 484%. Consequently, unless stated otherwise, we limit the cardinality of kterms in all future experiments to 3.

5.4 Constant Time and Space

FSD computes a document's degree of novelty with respect to all documents that arrived before. Since FSD systems operate on data streams, the set of previous documents is constantly growing. Each additional document needs to be stored and con-

Cardinality	weighted combined C_{min}	runtime (sec)	number of kterms formed
1	0.9822	0.328	1,715,007
1-2	0.8803	3.083	16,104,848
1-3	0.7902	19.389	101,273,711
1-4	0.7893	93.884	490,368,152
1-5	0.7891	373.632	1,951,513,245

Table 5.6: compares the detection cost (C_{min}) and efficiency (runtime in seconds) for combined cardinalities of kterms

sidered during future novelty computations. To remain feasible, an FSD system is required to compute a document's novelty in constant time and space $(O(1))$, independently from the number of previously arrived documents.

Constant Time

On the arrival of a new document, kterm hashing creates its content - the set of all kterms. Document novelty is computed based on the content novelty, which requires determining the membership of each kterm in the memory by hashing it onto its bit array and a subsequent lookup, whether the corresponding bit is set. Consequently, kterm hashing operates independently from the number of previously processed documents, as all computational steps only depend on the number of kterms formed. Twitter's 140 character limit limits the number of possible terms to 70 - blanks are considered in Twitter's 140 character limit. As a result, the time complexity of kterm hashing is $O(1)$ with respect to the number of documents processed.

If documents are not limited by the number of words, they need to be reduced to the top k weighted terms, as was usual during the TDT competitions (Yange, et al., 1998; Yamron et al., 1998). Figure 5.4 shows the throughput measured by tweets per second, as we process more and more tweets. The throughput curve initially increases until 20,000 documents, after which it flattens out and remains constant. The set-up of the bit array causes an initial overhead that explains why the throughput increases. Once the overhead is compensated, the curve flattens out and kterm reaches maximum throughput at 7,000 tweets per second. A more comprehensive efficiency experiments and comparison with other state-of-the-art systems appears in Section 5.5.

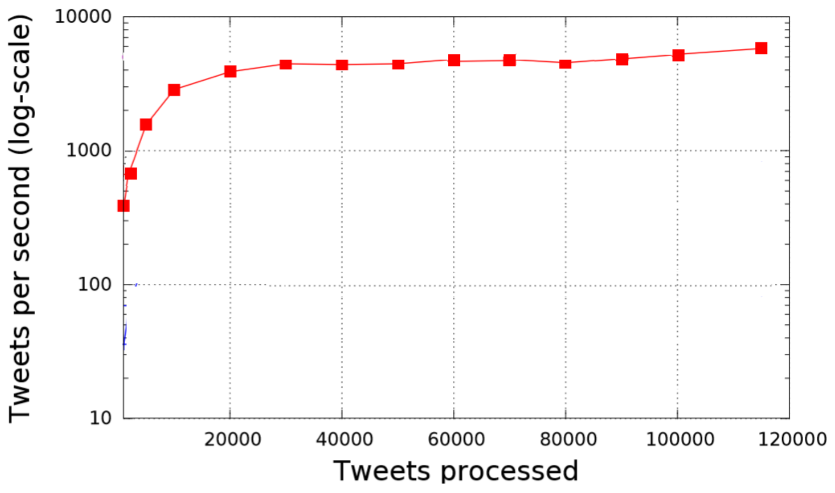


Figure 5.4: Illustration of constant throughput (tweets per second) for kterm hashing when processing more and more documents.

Constant Space

Kterm hashing relies on a memory that maintains the information of all previously encountered kterms. By Heaps law (Egghe, 2007) the number of distinct words - and subsequently the number of k-terms, grows continuously. Figure 5.5 shows the growth of all kterms formed from the 115,000 documents of Cross-Twitter 115k, which need to be stored in the memory for future novelty computations. The plot additionally shows that growth of unique kterms, those previously unseen with respect to the path. Every new kterm added to the memory grows its size. Although memory size increases only gradually (each stored kterm has a memory-footprint of 1 bit), kterm hashing needs to address this issue when operating on unbounded data streams. To ensure operation in constant space, we restrict the memory to a fixed length bit array, as described in Chapter 4. The length of the bit array determines the maximum size of the memory. Limiting the memory size to a predefined and fixed magnitude guarantees operation with constant space. Unfortunately, continuous insertions of newly encountered kterms into a fixed space will eventually saturate it. Increased space saturation negatively impacts detection accuracy. We explore the impact of space saturation on detection accuracy in detail in Chapter 7. To mitigate this problem, we introduce deletion strategies that ensure a constant load factor. The load factor corresponds to the fraction of bits set $(\frac{\text{number of set bit}}{\text{number of all bits}})$. Placing an upper limit on the load factor enforces a maximum ratio

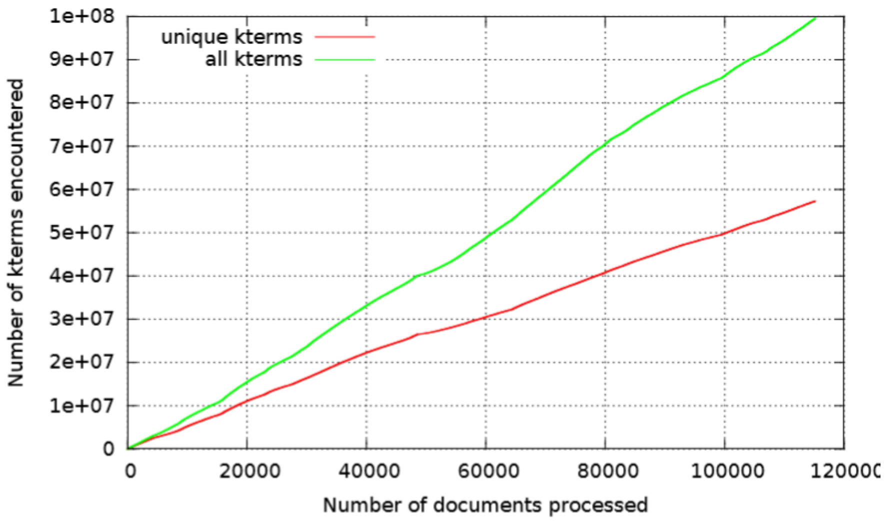


Figure 5.5: Illustration of the growth in the number of kterms encountered when processing Cross-Twitter 115k.

of set to unset bits, which ensures constant space.

In particular, we compare three deletion strategies: random deletion, temporal deletion and biased Reservoir Sampling. Whenever the load factor exceeds the pre-determined threshold ϕ, the deletion strategy zeros out a bit in the memory (bit array) to make room for new entries. This allows maintaining constant space at the cost of a small false positive rate.

A popular deletion strategies for streaming IR applications include deletion of the least frequent recent hitters (Berinde et al., 2009), as well as deletion according to uniform Reservoir Sampling (Vitter, 1985). Removal of the least frequent recent hitters requires maintaining a dedicated counter for each kterms encountered during the last time interval. Upon deletion, the bit with the least hits is zeroed out to make room for new entries. Identifying the lowest counter value requires a linear search through all counters, which is time-consuming. Although frequently used in other streaming applications (Muthukrishnan, 2005), we decided against this deletion strategy, because it requires storing and maintaining a large number of counters (1 for each recently visited bit), which negatively impacts the cost of time and space required to calculated a document's novelty. Additionally, one has to ensure that the counters operate in constant space to fulfil the stream requirements. Since FSD necessitates high efficiency, we decided against this deletion strategy following its increased space and time consumption.

Temporal deletion provides a more cost effective alternative to the deletion of the least frequent hitters. Instead of maintaining a precise counter for each recently processed kterm, temporal deletion only remembers the order in which kterms were added to the memory. Upon deletion, the bit associated with the oldest kterm is zeroed out. The kterm order is maintained in a queue, which is more efficient than maintaining a dedicated counter for each recently seen kterm. When looking up a kterm in the memory, the queue is searched and the corresponding entry is added to the end of the queue. When the maximum load factor is reached the bit in the memory corresponding to the first element in the queue is set to null and removed from the queue. This bit represents the kterm that has not been encountered the longest. Table 5.7 shows that temporal deletion results in the lowest increase in detection cost (C_{min}). However, the low increase comes at the expense of the highest increase (+126.2%) in runtime. All deletion strategies increase the runtime of FSD using kterm hashing. This is interesting since document deletion is one of the most effective speed up mechanisms for comparison based FSD approaches. The time complexity of those approaches directly depends on the number of documents stored. By contrast, the runtime of kterm hashing is determined by the number of kterms hashed onto the memory. Deletion strategies like temporal deletion reset an existing bit for each newly set bit, whenever the maximum load factor is reached. This requires an additional interaction with the memory, which doubles the runtime.

Reservoir sampling is frequently applied in streaming Natural Language Processing tasks (Shrimpton et al., 2015). The term “Reservoir Sampling” usually refers to *Uniform* Reservoir Sampling, which allows maintaining a sample of kterms, where each kterm formed from all documents in the stream is equally like to be a member of the sample. Kterms are added to the sample with probability $P = \frac{s}{n}$, whereas s denotes the sample size and n the number of kterms already added. The probability of new kterms being added to the memory decreases with each additional kterm added. When a kterm is inserted, an old kterm is removed at random to ensure a constant sample size. The resulting memory contains a set of kterms, whereas each encountered kterm is equally likely to be a member of the set. Although this property is useful for various NLP task, we decided against Uniform Reservoir Sampling. On social media stream, topics emerge, evolve and fade away (Petrovic et al. 2013). Consequently, the most similar previous document is more likely to be among the most recent documents which are less likely to be selected by Uniform Reservoir Sampling.

Biased Reservoir Sampling (Aggarwal, 2006; Osborne et al., 2014) is a variant of Uniform Reservoir Sampling that allows moving the selection bias either towards more recent or more distant kterms. In contrast to Uniform Reservoir Sampling, new kterms are added with a fixed probability (P), and insertions trigger random deletions. We apply Biased Reservoir Sampling while shifting the bias towards more recent documents by increasing the probability (P) of adding kterms to the memory. As a result, most of the recent kterms are considered by the memory and occasionally some older kterms are preserved as well. Biased Reservoir Sampling allows mimicking random deletion strategy by setting the insertion probability to 100% ($P = 1$). Random deletion adds all newly encountered kterms to its memory and deletes an existing entry at random. When repeatedly selecting a bit with uniform likelihood to be zeroed out, older bits are more likely to be reset. In contrast, uniform Reservoir Sampling with $(P < 1)$ does not add all kterms to the memory. Table 5.7 shows that for all tested versions of Biased Reservoir Sampling, FSD using kterm hashing resulted in the lowest increase in detection cost (+0.3%) when applying a random deletion strategy ($P = 1$). Decreasing the probability P of adding new kterms to the memory shifts the bias towards temporally more distant documents, which impacts FSD accuracy negatively. This result confirms Yang et al. (1998) and Petrovic et al. (2013), who measured increased FSD accuracy when focusing novelty detection on recently encountered documents. We further explore the impact of recency on detection accuracy in Chapter 6. Whenever the maximum load factor is reached, Random Deletion resets an existing bit for each new entry. This requires consulting the memory twice for each kterm added, which doubles runtime (+106.9%). Decreasing the insertion probability (P) reduces the additional effort at the expense of increased detection cost.

Figure 5.6 illustrates the memory footprint of FSD using kterm hashing with random deletion when processing more and more tweets. Kterm hashing with a deletion strategy operates with a strictly constant memory footprint independent of the number of documents processed. A constant memory footprint in combination with a constant processing time independently from the number of documents processed fulfil the streaming requirements of (Muthukrishnan, 2005) (O(1)) and allow kterm hashing to scale to unbounded data sets.

Algorithm	C_{min}	**Difference**	**Runtime (sec)**	**Difference**	**constant space**
no deletion strategy	**0.7966**	-	19.389	-	NO
temporal deletion	0.7979	+0.2%	43.854	+ 126.2%	YES
random deletion / Biased Reservoir Sampling (P = 1)	0.7986	+0.3%	40.125	+ 106.9%	YES
Biased Reservoir Sampling (P = 0.75)	0.8343	+5%	30.824	+ 59%	YES
Biased Reservoir Sampling (P = 0.5)	0.8832	+10.9%	21.029	+ 8.5%	YES
Biased Reservoir Sampling (P = 0.25)	0.9382	+17.8%	**10.824**	-44.2%	YES

Table 5.7: showing the impact of different deletion strategies on the effectiveness of FSD using kterm hashing on Cross-Twitter 115k

5.5 Difference to comparison based FSD

In the following section we compare the performance of FSD through kterm hashing with traditional comparison based methods. Our Baselines, which were described in detail in Chapter 3, consist of UMass, a state-of-the-art high precision FSD system and LSH-FSD the currently most efficient FSD system that also reaches state-of-the-art accuracy. We base our comparison on effectiveness, measured by the minimum normalized detection cost (C_{min}) and efficiency, measured by the throughput ($\frac{tweets}{second}$). Since effectiveness and efficiency can be traded off against each other, its important to examine them in combination. For example, the most efficient FSD system assigns a “first story” or “follow-up” label to all documents, which depending on the hardware, reaches a throughput of about 4bn tweets/second. The highest possible efficiency also comes at the highest possible cost in effectiveness ($C_{min} = 1$).

Algorithms like LSH-FSD and kterm hashing provide numerous parameters that allow tuning the trade off between effectiveness and efficiency. To ensure a fair comparison, we define a target effectiveness determined by the detection accuracy of UMass. The UMass FSD system is widely known as one of the most accurate FSD systems avail-

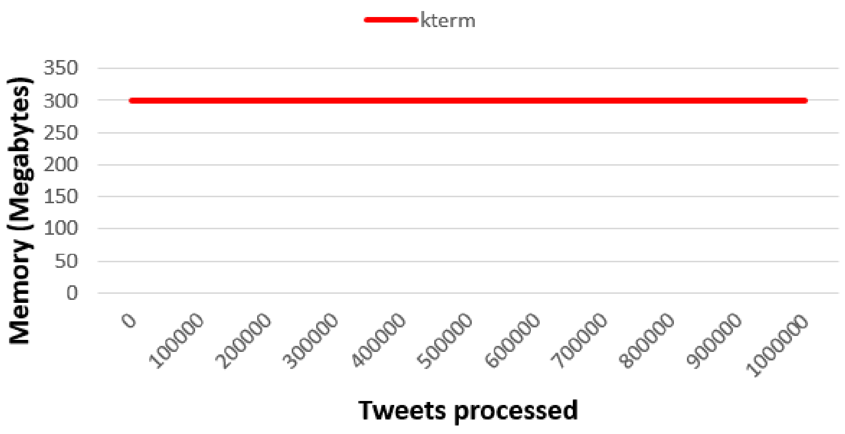


Figure 5.6: Illustration of constant memory footprint (measured in megabytes) for kterm hashing when processing more and more documents.

able (Petrovic et al., 2013; Moran et al., 2016). Given a target for effectiveness, we adjust the parameters of LSH-FSD and kterm for maximum efficiency while retaining an accuracy comparable to those of UMass.

Before we compare the performance of the three algorithms using single-digit metrics, we look at their DET plot, seen in Figure 5.7. DET plots provide a comprehensive comparison of detection accuracy because they show the performance for the full range of parameters. An introduction on DET plots appeared in Chapter 1. Figure 5.7 shows the efficiency tradeoff between false alarm and miss probability for UMass, LSH-FSD and kterm. All three systems perform consistently better than random performance, indicated by the red solid line. UMass outperforms LSH-FSD and kterm in the high precision area, where the false alarm probability falls below 2%. All other regions show comparable accuracy for all three algorithms. The DET plot in Figure 5.7 additionally provides the performance of UMass by its 90% confidence interval, illustrated by the two solid black lines to the right and left of UMass's line. These two lines form a confidence corridor, whereas all points that fall within the corridor are statistically indistinguishable from UMass. The DET plot shows that the accuracy of LSH-FSD and kterm is on par with UMass, a state-of-the-art high accuracy FSD system. This ensures that all three algorithms meet our target accuracy (defined in the beginning of Section 5.5), which makes them comparable in terms of efficiency.

Table 5.8 compares kterm with the two baselines in terms of (C_{min}) and throughput

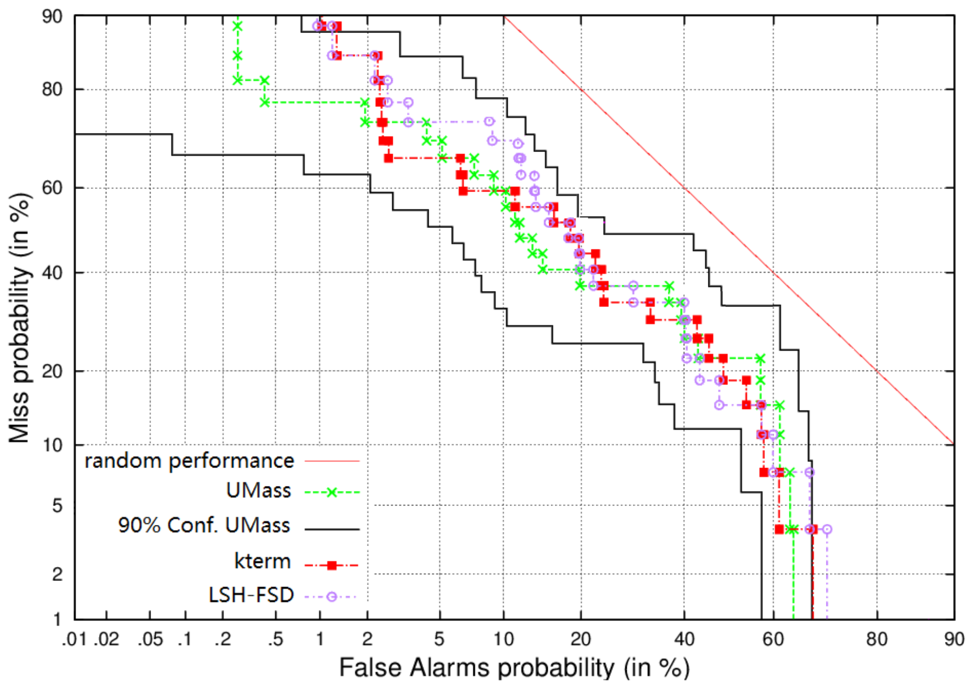


Figure 5.7: DET plot of UMass, LSH-FSD and kterm, showing that LSH-FSD and kterm are statistically indistinguishable from UMass in terms of detection accuracy.

measured on Cross-Twitter 115k. In addition to basic kterm, we also compare kterm with different biases for reservoir sampling. UMass shows state-of-the-art accuracy (Cmin = 0.79, lower is better), but can only process 30 tweets per second. Note, that UMass becomes progressively slower as each additional documents processed increases the computation effort for all future documents. LSH-FSD applies sophisticated document pruning and consequently operates 17 times faster at the cost of a 13% decrease in accuracy (Cmin = 0.90). Remember the difference in accuracy between LSH-FSD and UMasss is not significant. Our system (k-term) operates on par with UMass in terms of accuracy while being 2 order of magnitude (197 times) faster. Even in comparison with LSH-FSD, the currently fastest FSD system, kterm performs 1 order of magnitude (11.8 times) faster. The table performs 7.8 times faster than LSH-FSD when restricted in space.

5.5.0.1 Constant Time and Space

In addition to effectiveness and efficiency, we compare kterm to LSH-FSD and UMass in terms of their space and time requirements when processing document streams.

Algorithm	**Cmin**	**Difference**	**tweets/sec**	**speed-up**
UMass	0.7981	-	30	-
LSH-FSD	0.9061	-13.5%	500	17x
k-term no deletion	**0.7966**	+0.2%	**5,900**	197x
k-term biased reservoir sampling (P=01)	0.7986	-0.1%	2,900	97x
k-term biased reservoir sampling (P=0.75)	0.8343	-4.5%	3,700	123x

Table 5.8: Comparing the effectiveness and efficiency of different deletion strategies for k-term hashing with UMass and LSH-FSD

Operation with constant time and space with respect to the number of documents processed is a core requirement for algorithms to remain feasible in a true streaming environment.

Constant Time

In our first experiment, we apply UMass, LSH-FSD and kterm to Cross-Twitter 115k and measure the throughput (tweets per second) as they process more and more documents. Figure 5.8 provides the throughput curves of the three algorithms, which illustrate constant throughput for LSH-FSD and kterm. As described in Section 5.4, kterm and LSH-FSD require an initialization phase, which explains the lower throughput during the first 10k documents. Both algorithms amortize their initialization cost at about 25k documents, after which they operate with a constant throughput. The curve of UMass shows an opposite pattern, as it starts with the highest throughput of all three system at 1000 documents per second. The initially high throughput erodes quickly under 100 tweets per second after processing 10.000 documents. UMass is a comparison based FSD system without deletion. This enables it to deliver state-of-the-art accuracy at the cost of continuously decreasing throughput, as seen in Figure 5.8. Each incoming document is compared to all previously arrived documents if they share at least 1 term. The number of matching documents is initially low, which explains the high throughput at the beginning of the stream. However, each additional document processed increases the computational cost of future documents and causes the throughput to continuously decrease.

Figure 5.8 additionally shows the average output of the full Twitter stream - Firehose[2] at 5,787 tweets per second, illustrated by the solid magenta line.

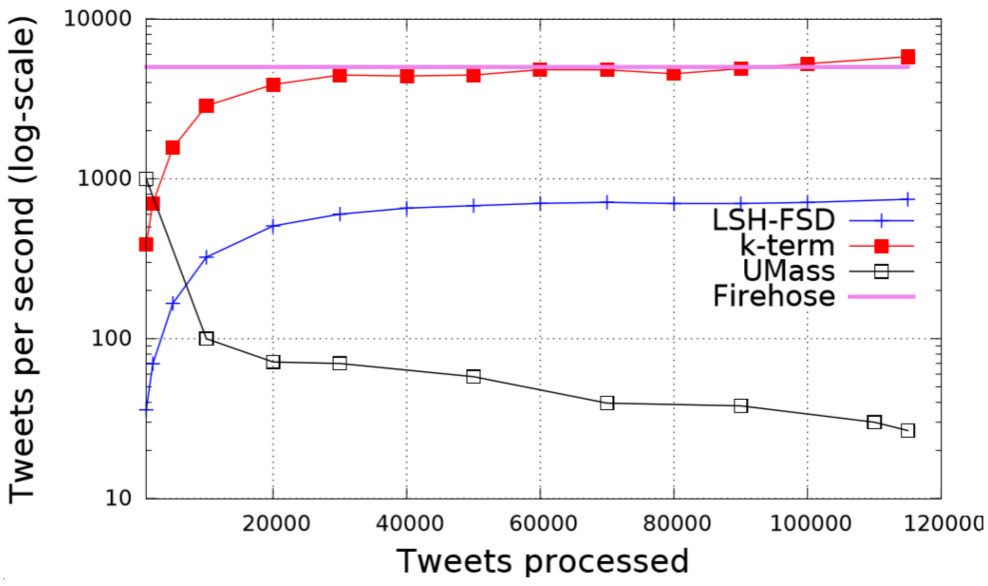


Figure 5.8: Illustration of constant processing time for LSH-FSD and kterm as well as continuously decreasing throughput for UMass.

Interestingly, kterm reaches the equivalent of the average Twitter Firehose stream on a single Intel-Xeon CPU core with 2.27GHz. Note, that this is very modest hardware and kterm processes more than 7k tweets per second if deployed on modern hardware equivalent to 2.2 GHz Intel Core i7-4702HQ. By contrast, the currently fastest FSD system - LSH-FSD requires a cluster of 72 cores (McCreadie et al., 2013) to reach 4,518 tweets per second.

Constant Space

Our second experiment investigates the memory requirements of our algorithm (kterm) in comparison to our baselines. We measure the memory footprint of the process in megabyte as they process more and more documents. Figure 5.9 provides the memory requirement for kterm and LSH-FSD. The memory footprint of the UMass FSD

[2]https://about.twitter.com/company (last updated: March 31, 2016)

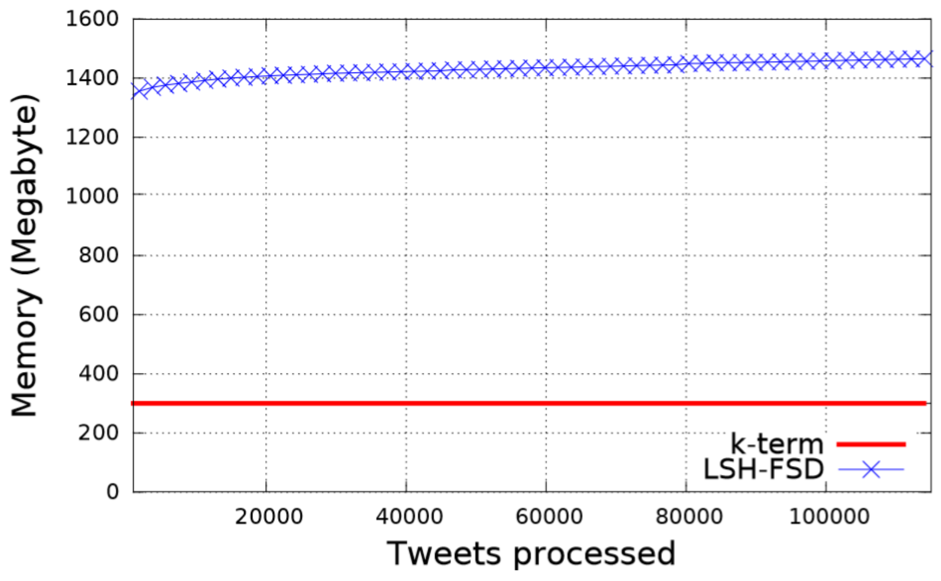


Figure 5.9: Illustration of constant space requirement of kterm and slightly but steadily increasing memory footprint of LSH-FSD.

system grows linearly with the number of documents processed and is therefore not considered for this experiment. Petrovic et al. (2010) and Petrovic et al. (2013) claim that LSH-FSD operates with constant space with respect to the number of documents processed. However, we measure a slight but steady increase in memory footprint with every new document processed. LSH-FSD uses hyperplanes to segment the vector space and form hash buckets. By Heaps law (Egghe, 2007) the number of distinct words - and subsequently the size of the hyperplanes, continuously grow. New terms added to the hyperplanes and inverted index explain the observed memory footprint growth of LSH-FSD. Figure 5.9 also shows that kterm operates with absolute constant space.

5.6 Conclusion

In this Chapter, we applied kterm hashing to FSD and explored the impact of preprocessing and kterm cardinality on detection accuracy. This chapter also compared FSD using kterm hashing to two strong baselines and found that all three reach state-of-the-art accuracy. While the three systems performed on par in terms of accuracy, kterm hashing demonstrated throughput up to two order of magnitude faster. Additionally, we investigated the time and space requirements of LSH-FSD and kterm hashing. Although LSH-FSD is supposed to perform in constant space, we measured a slightly but steadily increasing memory footprint. Kterm, by contrast, operates with absolute constant time and space.

Chapter 6

The Effect of Recency on FSD Detection Accuracy

6.1 Motivation

On social media and news streams, new topics continuously emerge, evolve and eventually fade away. Documents of a certain topic are assumed to occur in clumps i.e. they are considered to arrive with close temporal proximity. Yang et al. (1998) discovered that the average topic duration on news-wire data sets of TDT is about 2 month. They hypothesized that recently encountered documents are better suited to determine a new document's novelty than older documents. Consequently, they explored the concept of recency to boost detection performance.

In this chapter, we review the use of recency in the Yang's CMU-FSD system and show that LSH-FSD also considers a temporal aspect through variance reduction. Additionally, we are the first to uncover that the UMass FSD system is also biased towards more recent documents. We explore the impact of UMass's unique temporal bias and its impact on detection accuracy. We also show how recency is applicable to memory-based novelty detection. Instead of placing the bias on the temporal proximity of documents, we retain a queue of the most recently encountered kterms and inflate the novelty score of new documents, whose kterms match with it. Our experiments show that biasing novelty detection towards recency decreases the normalized minimum detection cost of kterm hashing by 2%.

6.2 Recency in CMU and LSH-FSD

Yang et al. (1998) reported consistently better accuracy for the CMU-FSD system when prioritizing recent documents for current decision making. They normalized the similarity of the current document to the closest previous document by the temporal difference between them, as seen in Equation 6.1.

$$novelty(d_n) = 1 - \max_{d_i \in window} \{cosine\ similarity(d_n, d_i) * \frac{i}{|window|}\} \qquad (6.1)$$

Remember that instead of computing novelty with respect to all previous documents, CMU retains a window covering the k most recent documents. Consequently, documents with increasing temporal proximity have progressively less influence on current novelty scores.

Petrovic et al. (2010) applied a conceptually related method to their LSH-FSD system by incorporating an exhaustive back-off step, which they call “variance-reduction”. They report that LSH alone yields poor detection performance because LSH occasionally fails to identify to closest previous document if it is not sufficiently close to the new document. Therefore, they additionally search the k most recent documents exhaustively, whenever LSH fails to provide a sufficiently close document. In addition to reducing the variance of pure LSH, the variance-reduction strategy also prioritizes recent documents, which are exhaustively searched whenever LSH fails to provide a sufficiently similar document. Consequently, instead of discovering the true nearest document in the past, LSH-FSD chooses to determine novelty based on the closest most recent document.

6.3 UMass and the Temporal Bias

The UMass FSD system performed consistently better than CMU-FSD and was used as a high effectiveness benchmark by LSH-FSD, which matches its detection accuracy. Interestingly, none of the publications describing the UMass FSD system reports about harnessing recency to boost detection performance. According to (Allan et al., 2000) UMass solely bases its decision on the inverse of the cosine similarity to the closest previous documents. However, when computing the cosine similarity of each document with its closest previous documents, which mimics UMass description, we encountered a divergence from the cosine similarity (1 - novelty score) obtained from UMass. Figure 6.1 shows a plot comparing the actual cosine similarity and the cosine similarity according to UMass.

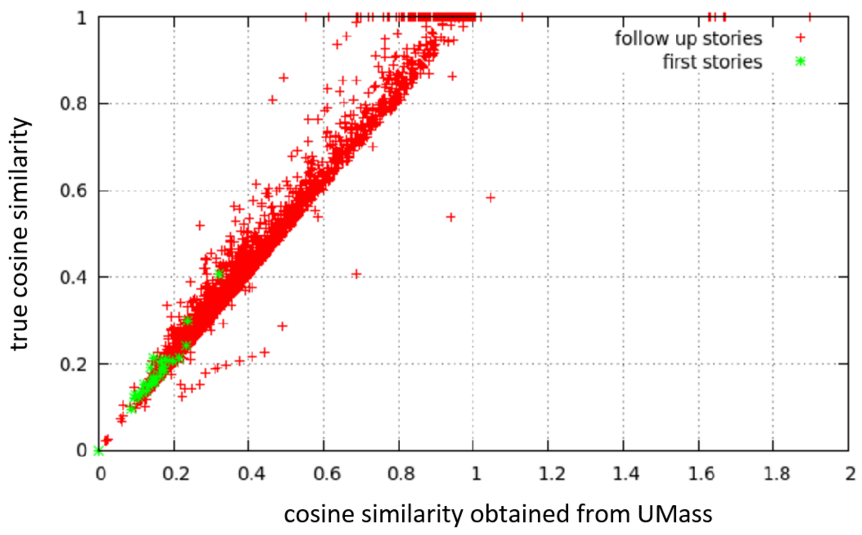


Figure 6.1: Illustration of the novelty score obtained from UMass and our re-implementation using pure cosine similarity.

As expected, the actual cosine similarity varies between 0 and 1. A cosine similarity equal to 0 indicates that a document does not share any terms with any previous document. By contrast, a cosine similarity of 1 results from a document that is identical to a previous document. Interestingly, UMass produces cosine similarities that exceed 1, which is impossible by the definition of the cosine similarity, given by Equation 2.4. While examining the original source code of UMass, which was made available to us, we found that the inflated cosine similarity scores originate from a unique way of handling incremental term statistics. We decided to elaborate on this discovery in detail since it substantially boosts detection accuracy and has not been made public yet.

6.3.1 Recency for through Cumulative Term Statistics

The UMass system calculates a document's novelty based on the inverse $tf.idf$ weighted cosine similarity to the closest previously encountered document. We already introduced the $tf.idf$ weighting scheme in Chapter 2, and mentioned that the idf component, seen in Equation 6.2, computes how representative a term is for a document.

$$idf(t) = log(\frac{|D|}{|\{d : t \in d, d \in D\}|}) \tag{6.2}$$

In a nutshell, idf assumes that terms that appear in only a few other documents are more descriptive than terms that are commonly used in many documents. According to Equation 6.2, the idf values of a term t depends on the proportion between the number of documents ($|D$) and the number of documents containing term t.

The $tf.idf$ term weight is the product of the idf component and the number of times the term occurs in the document. Cosine similarity based novelty computation uses the $tf.idf$ weights for calculating the dot product and vector lengths, as seen in Equation 6.3.

$$Cosine\ similarity(v_1, v_2) = \frac{dot\ product(v_1, v_2)}{length(v_1) * length(v_2)} \tag{6.3}$$

FSD is a streaming task. Documents arrive one-at-a-time and information about how many future documents contain a specific term is not available. Following the absence of global term statistics, UMass approximates idf components by computing them incrementally. On the arrival of a new document, its terms are added to the inverted index and their corresponding idf values are updated. Consequently, incremental idf statistics are based on all documents processed in the past. Incremental document statistics are not uncommon in streaming applications and they are also applied by LSH-FSD.

Close inspection of the UMass source code revealed that the idf weights are incrementally computed as expected. However, UMass does not pass the updated idf scores to the length normalisation of the cosine similarity. Instead of recomputing the document vector length using the updated idf values, UMass keeps the length that was computed at the arrival of each document. Consequently, UMass computes novelty according to Equation 6.3, whereas the dot product uses the incrementally updated idf components but the vector length of the previous vector v_2 is *not* recomputed based on the updated idf components. We assume that the update of the vector length was skipped to increase efficiency, unaware of the impact it has on detection accuracy.

To outline the effect of skipping the length update, we force UMass to re-compute the vector lengths using the updated incremental idf scores.

UMass	C_{min}	Difference
with vector length update	0.8715	-
without vector length update	0.7981	-9.2%

Table 6.1: impact of vector length update on the detection accuracy of UMass

Table 6.1 illustrates that skipping the update of the vector length not only increases the efficiency of UMass but also substantially reduces the normalized minimum detection cost by 9%.

We claim that skipping the length update biases novelty computation using cosine similarity towards more recent documents, which explains the decreased detection cost. Our claim is based on the fact that when processing a stream, the number of encountered documents rises faster than the number of documents that contain a specific term, because most terms are not present in all documents. Consequently, idf components increase over time, which inflates the $tf.idf$ weights of terms in documents later on in the stream. Inflated $tf.idf$ components increase vector lengths, which are used by the cosine similarity to squash the dot product, as seen in Equation 6.3. As a result dot products of documents with little temporal proximity are less squashed than those of greater temporal proximity. Since novelty is by definition 1 minus the similarity, increasing the similarity of temporally distant documents decreases their novelty. We conclude that skipping the vector length update biases novelty detection towards more recent documents, which increases detection accuracy.

For example, in Cross-Twitter 115k document number 10,067 is a first story, whose most similar previous document (document number 286) arrived 9,781 documents before it. The novelty score using the correct cosine similarity for document 10,067 is 0.4071. Based on the temporal distance, UMass inflates its cosine similarity and assigns a novelty score of 0.3205.

6.4 Biasing Kterm Hashing Towards Recency

Biasing novelty detection towards more recent document has shown to be an effective way to boost detection performance of comparison based FSD systems including CMU-FSD, UMass and LSH-FSD. In the following section we explore the use of recency for kterm hashing.

In contrast to the previously named FSD systems, kterm hashing does not apply comparison based novelty computation. Therefore, kterm hashing does not maintain previously encountered documents and novelty is not based on 1 minus the document similarity. Instead, kterm hashing maintains the memory, a collection of previously encountered kterms and novelty is based on a document's proportion of unseen kterms to the number of all kterms. Consequently, kterm hashing does not allow maintaining a window of recent documents, whose novelty scores are deflated when compared with a new document, as done by CMU and LSH-FSD.

Instead, we retain queue Q, which holds the m most recently encountered kterms in their chronological order. Whenever a kterm k is formed that occurs in Q, the bias is increased based on the proportion of variable θ and the position of k in Q. We optimize θ and m on our separate training set, whereas best performance was achieved with $\theta = 4$ and $m = 3000$. Given this set-up, kterm hashing inflates the normalization of the content novelty when a document's kterm matches a recently encountered kterm. This deflates the document novelty and biases kterm hashing towards recency. Algorithm 6 provides the detailed pseudo code for kterm hashing with a recency bias.

kterm hashing	C_{min}	Difference
without recency bias	0.7966	-
with recency bias	0.7792	-2.2%

Table 6.2: Impact of biasing kterm hashing towards more recent documents on detection accuracy

Table 6.2 illustrates the impact of biasing kterm hashing towards recent documents on detection accuracy when applied on our test set, Cross-Twitter 115k. The recency bias for kterm hashing decreases the normalized minimum detection cost by 2.2%.

Algorithm 6 : kterm with bias on recency

```
1:  initialize memory $M$, queue $Q$
2:  for all document $d_n \in Stream$ do
3:      $c_n \leftarrow \{t : t \subset d_n, |t| \leq k\}$
4:      for all kterm $t \in c_n$ do
5:          if $t \in Q$ then
6:              $bias \leftarrow bias + \frac{\theta}{position\ of\ t\ in\ Q}$
7:          end
8:          if $t \notin M_{n-1}$ then
9:              $novelty_{c,n} \leftarrow novelty_{c,n} + 1 * \alpha_{|t|}$
10:             $M_n \leftarrow M_{n-1} \cup t$
11:             if $t \in Q$ then
12:                 $Q \rightarrow t$
13:             end
14:             $Q \leftarrow prepend(Q,t)$
15:             if $|Q| > m$ then
16:                 $Q \rightarrow last\ entry$
17:             end
18:         end
19:     end
20:     $novelty_{d,n} = novelty_{c,n} * (|c_n| + bias)^{-1}$
21: end
```

6.5 Conclusion

This chapter outlined the positive effect of biasing novelty detection towards recency on detection accuracy. We reviewed the recency bias of CMU and LSH-FSD and were the first to discover that UMass also considers recency by deflating the novelty scores of temporally distant documents. We discovered that the UMass system does not pass the incrementally updated idf values to the vector length of previous documents. We explained why neglecting the length update deflates the novelty scores of temporal distant documents, which decreases the normalized minimum detection cost by 9%. Eventually, we showed how to bias kterm hashing towards recency. Instead of documents, kterm hashing keeps a queue of the most recent encountered kterms. Documents, whose kterm match with a kterm in the queue, receive a novelty boost depending on the position of the kterm within the queue. Our experiments showed that the recency bias decreases the detection cost of kterm hashing by 2%.

Chapter 7

FSD Performance over time

7.1 Motivation

State-of-the-art FSD systems, like LSH-FSD, apply deletion strategies to limit the number of documents considered for novelty computation, which drastically reduces the computational effort and scales systems to high-volume data streams. Additionally, document deletion ensures operation with constant time and space, independently from the number of documents processed. In the following chapter, we study the impact of deletion on FSD performance and show that it also positively affects accuracy over time. To the best of our knowledge, no research up to this date showed the necessity of deletion by studying the effect of processing more and more documents on FSD accuracy over time. We show that when applying FSD systems without deletion strategy, like UMass, novelty scores decay over time because the more documents a system has processed the less likely it is to encounter new information. Novelty scores, usually bound between 0 and 1, reflect the degree of a document's novelty with respect to the past. We hypothesize that continuously adding document vectors to a finite space, gradually fills it up, which negatively impacts detection accuracy. Note that up to this point, deletion strategies are applied to scale FSD systems and to retain constant operation. High accuracy systems, like UMass, focus on effectiveness and don't apply deletion. Our experiments show that FSD without deletion results in continuously decaying novelty scores, which have a direct negative impact on FSD accuracy. The goal of our experiments is to reveal that deletion not only positively impacts efficiency and operation with constant space in time but also ensures constant accuracy.

Following the outcome of our experiments, we advocate that streaming applications like FSD, are required to retain a constant level of accuracy over time. When processing continues data sources, constant accuracy with respect to the number of documents processes is as critical as operation with constant time and space. If accuracy decays over time, an FSD system's effectiveness would continuously decrease and eventually render it useless. We therefore require streaming applications like FSD to operate in constant time and space while maintaining a constant level of accuracy over time.

Additionally, we introduce an alternative to deletion strategies applicable to FSD using kterm hashing. While studying the novelty decay of FSD systems without deletion, we find re-occurring decay patterns and derived decay models that allow approximating the expected novelty decay for any document position within the stream. Since our experiments show that novelty scores continuously decay, documents later in the stream are expected to have on average lower novelty scores than earlier documents. We show that using a decay model, we can counteract novelty decay for kterm hashing based FSD. This provides an alternative to deletion strategies for kterm hashing to ensure constant accuracy. Adjusting novelty scores according to the decay model also successfully prevents novelty score decay for comparison based systems like LSH-FSD and UMass. Unfortunately, they don't scale to a high number of documents without deletion.

7.2 Novelty Decay in Comparison based and memory-based FSD Systems

Our first experiments explore the impact of processing more and more data when no deletion strategy is applied, like for UMass. FSD systems process a stream of documents with a single pass over the data. For each arriving document a novelty score is computed, indicating its degree of novelty with respect to all previously seen documents. Figure 7.1 provides an example output of an FSD system. New events are detected whenever a document's novelty score, illustrated by the red dots, exceeds the predefined detection threshold, illustrated by the horizontal blue dashed line.

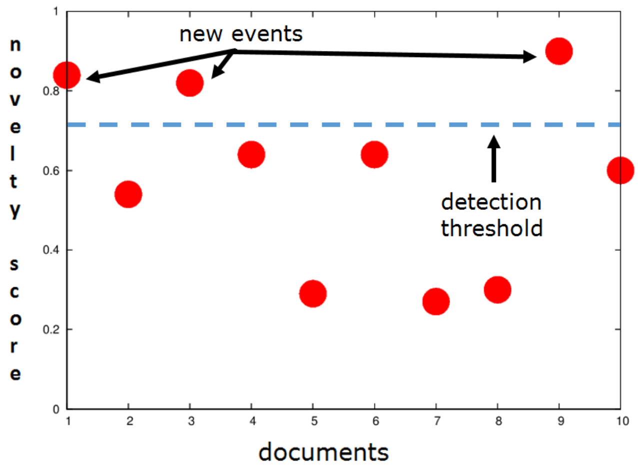


Figure 7.1: Illustration of the FSD principle. Whenever a document's novelty score surpasses the detection threshold, the document is considered to talk about a new event.

During the TDT program and for most other detection tasks, detection is based on a fixed threshold. The TDT competitions used data set in the range of several thousand documents, which was a reasonable size at the time. Figure 7.2 shows the cumulative average novelty score of UMass, a comparison based FSD system that does not apply any deletion, when processing 10,000 tweets. The number of tweets is comparable to the first TDT dataset. The graph illustrates that for such small data sets, a fixed threshold appears reasonable, as it mimics the behaviour of the average novelty score. During all TDT competitions the average novelty scores were not considered to drift over time and constant detection thresholds are applied by all FSD systems.

When TDT was introduced in early 2000, data streams and thus research data sets were of small scale. In contrast, today, 10k documents equate to less than 2 seconds of the Twitter stream. In our next experiment we investigate the impact of processing millions of documents in combination with a fixed thresholding strategy and no deletion on FSD performance.

Figure 7.3 shows the UMass novelty scores of the first 2 million documents of Twitter-Cross 52 mio. We limit the size of the dataset because the throughput of UMass steadily decreases and 2 million tweets is 2 order of magnitude large than the first TDT data set, which is sufficient to show the impact of processing more and more

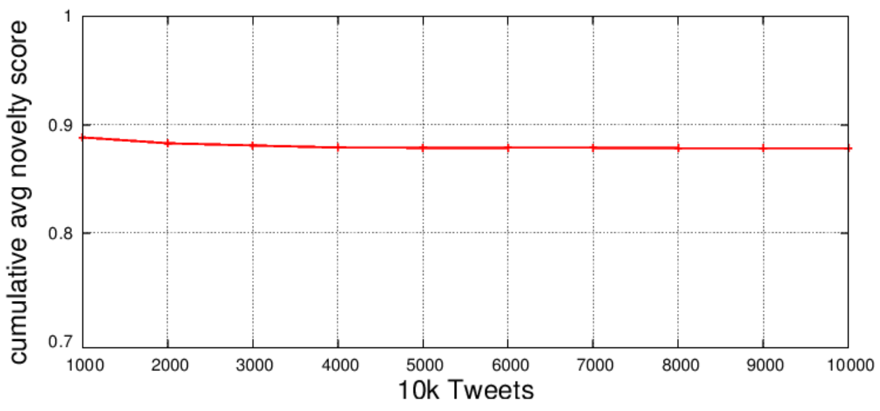


Figure 7.2: Illustration of the cumulative average novelty score of the UMass FSD system for 10k tweets.

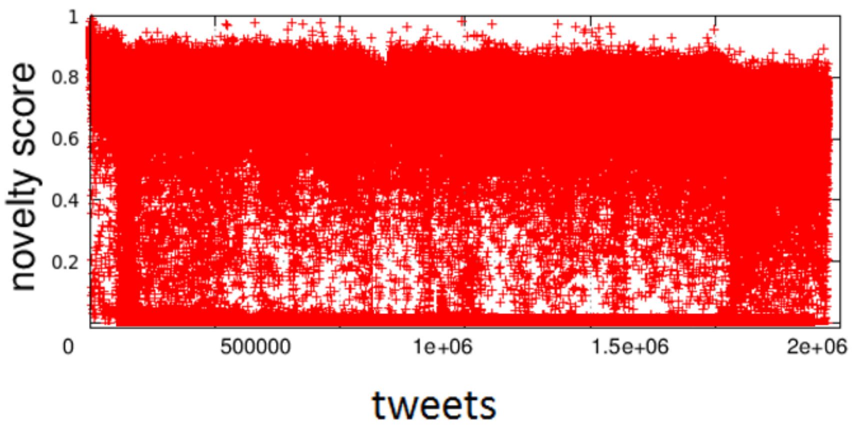


Figure 7.3: Illustration of the novelty scores from 2 million documents. The graph indicate that UMass detects "novel" documents during all positions in the 2 million document stream.

documents on FSD accuracy. The graph illustrates that UMass detects documents with high novelty scores at all position in the stream. We also see that UMass considers only documents in the very beginning as entirely novel. These documents have a novelty score of 1 and don't overlap with any previous document. This appears reasonable as with each additional document processed, the likelihood of encountering a document that does not share a single term with all previous documents, becomes progressively lower (Qin et al., 2017). To identify a trend in the novelty scores over time we compute their cumulative average and plot it against the document position.

Figure 7.4 shows the cumulative average novelty score of UMass when processing 2

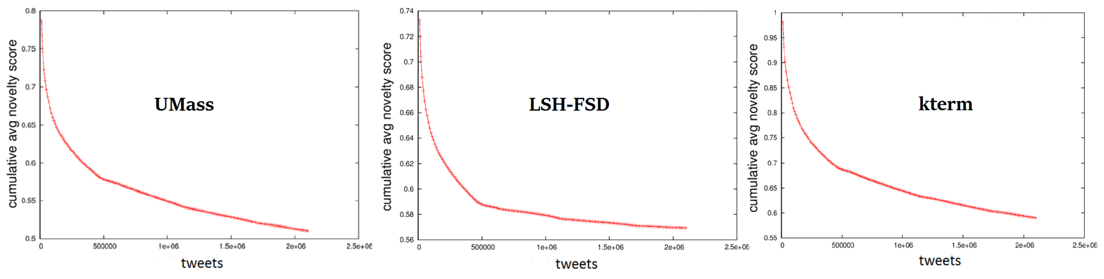


Figure 7.4: Illustration of the cumulative average novelty scores of a comparison, FSD and kterm based FSD system, when applied to 2 million tweets.

million tweets. Additionally, we present LSH-FSD and our kterm hashing based FSD system without deletion strategies to explore the affect of deletion strategies on the behaviour of average novelty scores. Note that the absolute novelty score values are not comparable between different FSD systems. LSH-FSD for example, produces lower average values than UMass or kterm. This does not indicate that LSH-FSD considers less documents as new events, as novelty is determined by an individually optimized detection threshold.

The curves in Figure 7.4 reveal a continuing decay of the average novelty score for comparison based and memory-based FSD systems without deletions, as they process more and more documents. In particular during the first 1 million documents, the average novelty score falls sharply.

This thesis is the first to reveal that the novelty score of comparison based FSD systems decay over time if no deletion strategy is applied. This decay has a direct impact on detection accuracy, which is based on constant thresholds, as seen in figure 7.5. The graph shows a severe drop in average novelty score, in particular during the first 1 mio documents. As a result, the average novelty score of documents in area A is above the optimal detection threshold, which indicates that many of them are likely considered to be “new events”. Consequently, this area increases the probability of false alarms (false positives). The average novelty score of documents in area B is below the optimal detection threshold. Documents in this region are likely being considered as “follow-up”. As novelty score are lower, new events in this area are more likely missed, causing false negatives.

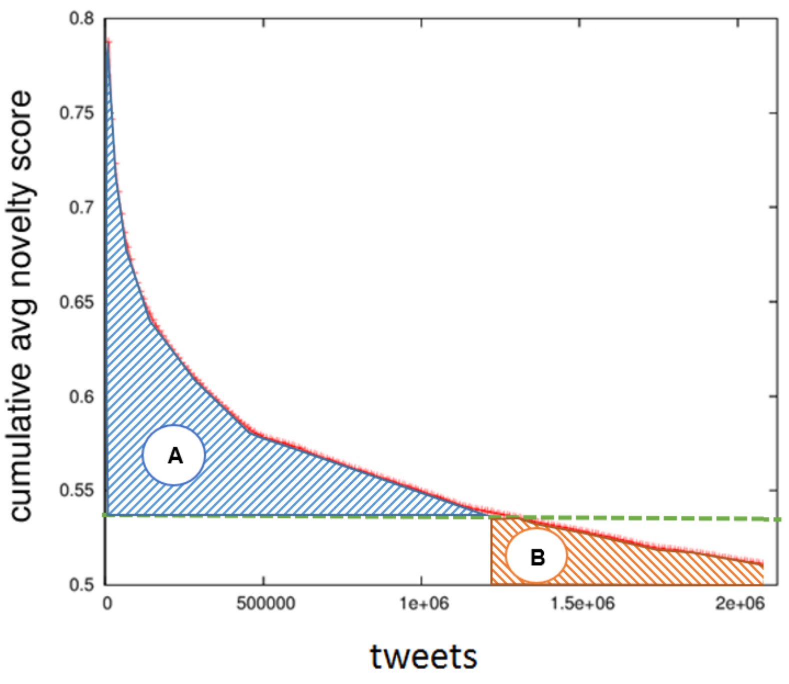


Figure 7.5: Illustration of the impact of novelty score decay on detection performance: showing that false positives are more likely in the beginning of the stream (area A) and false negatives are more likely later on (area B).

7.3 Causes for Novelty Score Decay over Time

Our experiment showed that comparison based and memory-based FSD systems suffer from a continuous decay in novelty scores when no deletion strategy is applied. We also explained that how decaying novelty scores impact detection accuracy negatively. To understand why novelty scores decay over time when no deletion strategy is applied, we explore the causes separately for comparison based and memory-based FSD.

Traditional Comparison based FSD

Comparison based FSD systems, like UMass, compare each new arriving document with all previously seen documents. Novelty scores depend on the distance to the closest previously encountered document. Figure 7.6 illustrates a simplified model of a 2-dimensional vector space. Hollow circles indicate previous documents. Full circles indicate a new document and the arrows show the distances between new documents and the closest previous document. An arrow's magnitude illustrates a document's degree of novelty. As more documents arrive, the space fills up. The more saturated a space becomes, the more likely it is that additional objects are close to existing ones.

We therefore hypothesize that the average novelty score declines with the increase in vector space saturation.

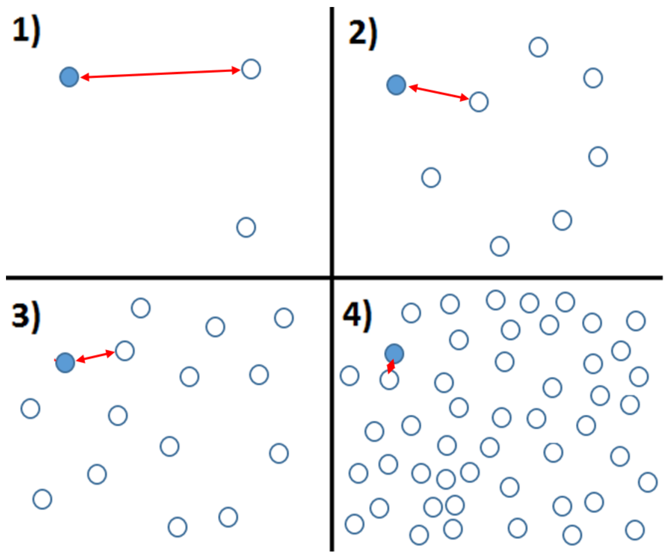


Figure 7.6: Illustration of a simplified model for a comparison based FSD system, like UMass, in a 2-dimensional vector space.

LSH based FSD

LSH based FSD systems, like LSH-FSD, share the basic concept of computing novelty with traditional comparison based systems. The advantage of LSH based systems resides in efficiency gains from limiting the search space from the entire vector space to the size of a hash bucket. Figure 7.7 illustrates a vector space separated by two hyper planes, forming 4 buckets. Although LSH reduces the amount of comparisons on average to $\frac{\#docs}{\#buckets}$ (the size of a bucket), saturation still grows with each new document added. As a result FSD using LSH without deletion suffers from the same novelty score decay as standard comparison based systems, as seen in figure 7.7.

memory-based FSD

Kterm hashing is a memory-based novelty detection method. Instead of comparing new documents to previous ones, all kterms of a document are formed and hashed onto the memory, which holds kterms from previous documents. The fraction of previously unseen kterms determines the degree of novelty. To keep track of past information, every document adds its own kterms to the memory. If no deletion strategy is in use, the memory fills up. The increasing saturation of the fixed sized memory resembles the saturating vector space of comparison based novelty detection and causes the average novelty scores to decay over time.

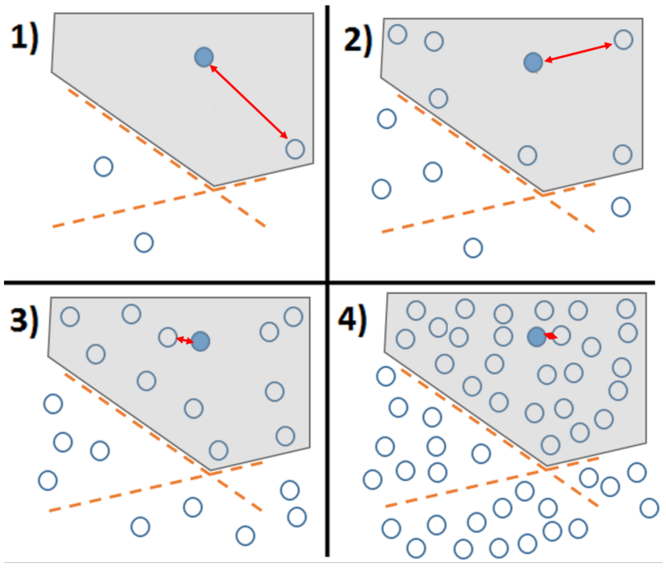


Figure 7.7: Illustration of a simplified model for an LSH based FSD system, like LSH-FSD, in a 2-dimensional vector space.

We hypothesize that novelty scores from comparison based and memory-based FSD without deletion steadily decay with the increase in space saturation. To verify our hypothesis we repeat the experiments above but this time we apply a deletion strategy to all three systems. Note that this experiment does not aim at achieving maximum detection accuracy but solely focuses on the behaviour of the observed cumulative average novelty score over time. The goal of this experiment is to show that even sub-optimal deletion strategies, like sliding windows, do not only positively impact efficiency and operation in constant time and space but also enable achieving constant accuracy.

Since UMass does not come with a deletion strategy, we apply a sliding window spanning 100.000 documents to it. Only documents inside the window are considered when computing the novelty score of a newly arriving document. New documents are added to the window until it fills up. Afterwards, for each new document added the oldest one falls out of scope. Note that UMass in conjunction with a sliding window closely resembles the CMU system described in Chapter 2. LSH-FSD additionally allows more sophisticated strategy based on local deletion, as described in Chapter 3. Instead of deleting (global) the oldest document whenever a new document arrives, LSH offers hash bin specific (local) deletion. Each hash bucket corresponds to a specific hyper-polygonal shaped region in the vector space. This allows placing an upper limit on the number of documents for each of them. Whenever a particular region in the vector space becomes too densely populated, one of its documents gets deleted. For FSD using kterm hashing, we apply its random deletion strategy. Instead of limiting the number of documents considered for novelty computation, deletion for kterm hashing limits the number of kterms in the memory by defining a maximum load factor. Documents contribute with their kterms to the memory until the maximum load factor is reached. Then, for each new kterm an old kterm, chosen at random, gets deleted. All three deletion strategies ensure that the space saturation remains constant independently from the number of documents processed.

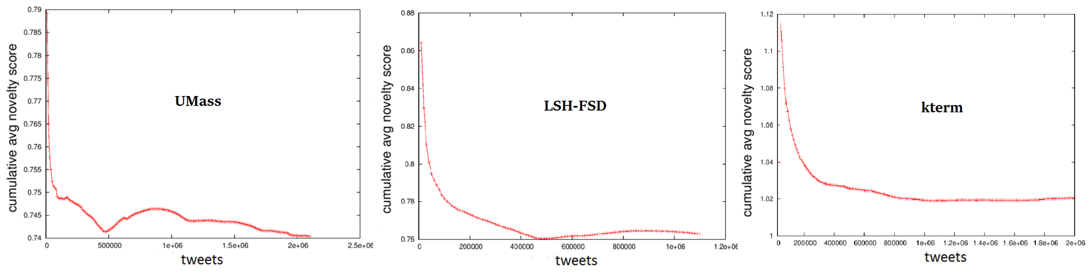


Figure 7.8: Illustration of the impact of a sliding window on the average cumulative novelty score.

Figure 7.8 shows the cumulative average novelty score when applying the deletion strategies to UMass, LSH-FSD and kterm hashing based FSD. The graph reveals that the novelty scores initially decrease up-until the window, bucket or memory is filled up. This is expected because during this stage, the system resembles operation without deletion. As soon as the maximum allowed space saturation is reached and deletion is applied, the cumulative average novelty score flattens out. All three deletion strategies show a similar impact on the average novelty score. Close inspection of the graphs shows a smoother flattening of the curve for LSH-FSD and kterm, which apply local deletion and kterm deletion respectively. By Contrast, the curve of UMass slightly bounces around a novelty score of 0.742. We conclude that deleting an individual kterm or a document of a particularly densely populated region (local deletion by LSH), is less likely to create sparse regions where new documents have higher average novelty scores. By contrast, sliding window based deletion removes documents according to their chronological order. Documents in social media streams that share temporal proximity are also likely to share topical proximity (Petrovic, 2013). We assume that removing topical clusters of temporal proximity creates sparse regions that cause bouncing average novelty scores.

We conclude that keeping the space saturation constant also ensures constant accuracy over time, which confirms our hypothesis. To the best of our knowledge, this is the first exploration of the impact of document deletion on average novelty scores over time.

Our experiments showed that FSD systems without a deletion strategy result in continuously decaying average novelty scores. We showed that novelty scores decay with the increase in space saturation. Consequently, the average novelty score remains constant when the vector space saturation remains constant. Keeping the number of documents constant is a widely applied principle in the FSD community, as described in Chapter 2. However, so far it has been applied as a means to speed up novelty computation, unaware of the positive impact it has on the accuracy over time.

7.4 Bootstrapping FSD Systems to Boost Detection Accuracy

This section provides a method for pre-initializing comparison based and memory-based FSD systems to boost detection accuracy. All FSD systems posses an internal representation of the past. This representation stores information of previously encountered documents and provides the basis for determining the novelty of future documents. UMass represents the past by a collection of document vector and incrementally accumulated idf statistics. LSH-FSD additionally holds hash-tables that store the associations of previous documents with their corresponding hash buckets and kterm hashing keeps kterms of previous documents in its memory.

When an FSD system is applied to a document stream, all these internal representations are empty, as FSD systems start in a NULL state. This state is similar to the "Cold Start Problem" (Lika et al., 2014) of recommendation system and explains why documents in the beginning of a stream are considered as first stories - they are new because there are no previous documents with which they could be compared to. Our experiments in Section 7.2 showed that the average novelty score sharply decreases during the first 1 million documents for comparison and memory-based FSD systems. The scores continue to decrease afterwards but at a lower rate.

We conclude that the average novelty scores drop in particular during the period from the 1^{st} to the $1,000,000^{th}$ document due to the cold start of FSD systems.

To overcome the cold start problem, we pre-initialize FSD systems with unrelated data in the form of 1 million random tweets that pre-date Cross-Twitter 115k. Processing these tweets initializes the internal states of the FSD systems, before they begin operation on the Cross Twitter 115k document stream. We found that bootstrapping FSD systems with unrelated data decreases the minimum detection cost by around 5%, as seen in Table 7.1. Note that for this experiment we run UMass, kterm and LSH-FSD with optimal settings using optimal deletion strategies. In order to demonstrate the impact of all three systems, we use Cross-Twitter 115k. It is important to ensure that the tweets for the pre-initialization step are random and unrelated tweets that pre-date the time period of Cross-Twitter 115k. Once the FSD systems process the first tweet of Cross-Twitter 115k, their inner states are not NULL, since they hold the information of 1 million random documents. Consequently, the first tweet of Cross-Twitter 115k is less likely to be entirely new to the FSD systems. Bootstrapping FSD systems by pre-initializing them with 1 million documents removes the sharp initial novelty decay and average novelty scores only decrease gradually, as seen in Figure 7.9. Avoiding the period of sharpest novelty score decrease improves detection accuracy but does not resolve the overall problem of novelty decay. Figure 7.9 illustrates that even without the sharp initial score decline, novelty scores continue to decrease.

Algorithm	C_{min} original	C_{min} prepended	Difference
UMass	0.7981	0.7646	-4%
LSH-FSD	0.9648	0.9166	-5%
kterm	0.7966	0.7584	-5%

Table 7.1: Impact on detection cost when appending 1 million documents to Cross-Twitter 115k

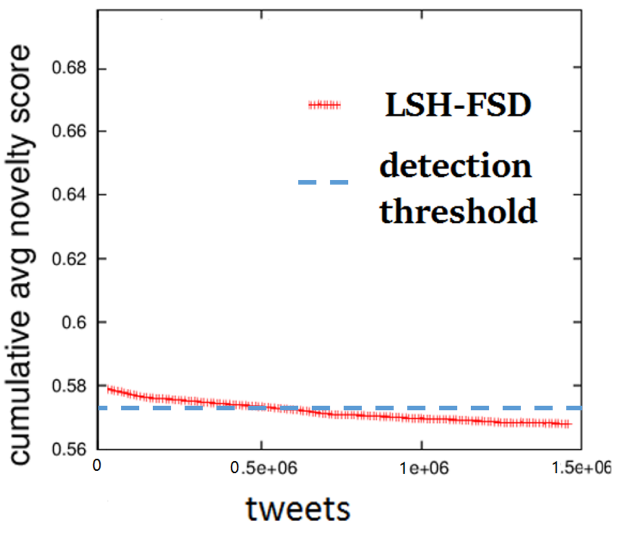


Figure 7.9: Illustration of the impact of bootstrapping an FSD system (kterm) by pre-initializing it with 1 million unrelated tweets on the average novelty score.

7.5 Additional Methods for Counteracting Novelty Score Decay over Time

Placing an upper limit on the space saturation successfully stops novelty decay over time for comparison based systems including UMass and LSH-FSD, as well as for memory-based systems, like kterm. Document deletion is essential for comparison based FSD system, like UMass and LSH-FSD, to achieve adequate throughput. In contrast, our experiments showed that document deletion decreases the throughput of FSD using kterm hashing. The time complexity of memory-based methods does not dependent on the number of documents processed but on the number of terms per document. Deleting strategies require resetting an existing kterm for each new kterm added to the memory. The additional effort hurts the throughput of FSD using kterm hashing. Applying kterm hashing without deletion causes the novelty scores to decay over time. In the following section, we provide an alternative method for kterm hashing that compensates novelty score decay without kterm deletion.

7.5.1 Modelling Novelty Score Decay

While studying the behaviour of cumulative average novelty scores over time for comparison based and memory-based FSD systems, we encountered reoccurring decay patterns. Our alternative to prevent novelty decay builds upon adapting novelty scores according to a document's position within the stream. Our experiments in Section 7.2 showed that documents that appear later on in the stream are more likely to receive low novelty scores, following the continues novelty score decay. We counteract the score decay by approximating the expected average novelty score decay, with respect to a particular point in time t within the stream. By t we denote the time-stamp, corresponding to a document's position within the stream. We hypothesize that modelling the expected decay at a particular point in time allows compensating it when computing novelty scores.

To derive a decay model, we apply kterm hashing to a test set, consisting of 52 million random tweets that pre-date those of Cross-Twitter. It is important to ensure that the 52 million tweets pre-date the test dataset to avoid that they contain information relevant to a target topic. We use the resulting cumulative average novelty scores to build our decay models. In particular, we apply logarithmic, exponential and polynomial regression to the observed cumulative average novelty scores of the 52 mio random tweets, while optimizing the coefficient determinant (R^2). The coefficient determinant indicates the proportional variance between approximated and observed cumulative average novelty scores.

Polynomial Model

Figure 7.10, 7.11, 7.12 illustrate the expected novelty score based on a polynomial function of the 4^{th}, 5^{th} and 6^{th} order (red dotted line) and compare it with the observed cumulative average novelty scores (blue solid line) from Cross-Twitter 52mio.

The red dotted line corresponds to the expected novelty score (EN) according to our polynomial decay model, shown in Equation 7.1. The equation provides an n^{th} order polynomial function that approximates the expected novelty score at a particular time t. The function's parameters ($\delta_0 - \delta_6$) were optimized on the test set consisting of 52 million random tweets.

$$EN(t) = \delta_6 * t^6 - \delta_5 * t^5 + \delta_4 * t^4 - \delta_3 * t^3 + \delta_2 * t^2 - \delta_1 * t + \delta_0 \tag{7.1}$$

Figure 7.10, 7.11, 7.12 also show the observed cumulative average novelty scores of FSD using kterm hashing on Cross-Twitter 52mio. Note that these are the novelty scores the decay models try to predict.

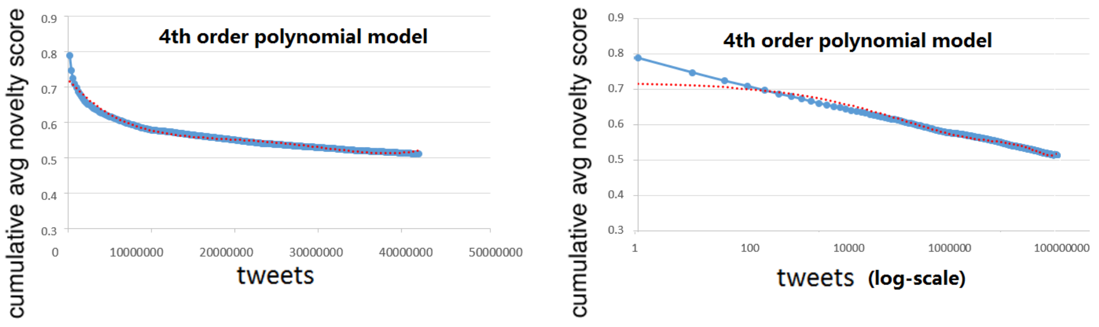


Figure 7.10: Illustration of novelty scores over time in linear and log-scale. The bold blue curve indicates the observed cumulative average novelty scores on Cross-Twitter. The red dotted line resembles the 4^{th} order polynomial model of the cumulative average novelty score from 52 million random tweets.

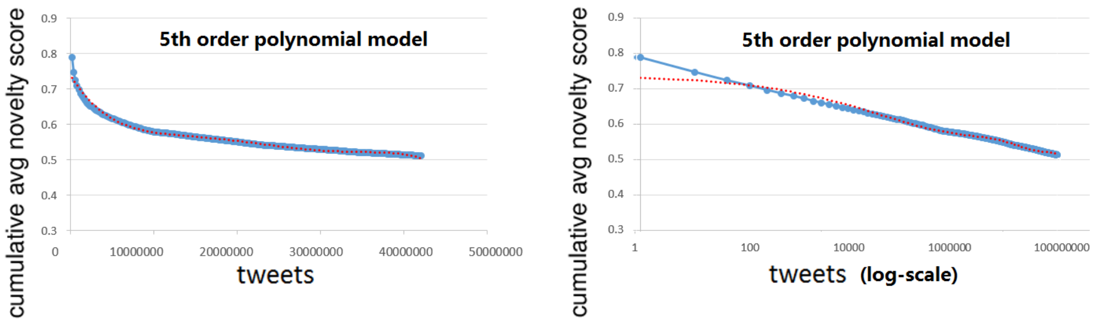


Figure 7.11: Illustration of novelty scores over time in linear and log-scale. The bold blue curve indicates the observed cumulative average novelty scores on Cross-Twitter. The Red dotted line resembles the 5^{th} order polynomial model of the cumulative average novelty score from 52 million random tweets.

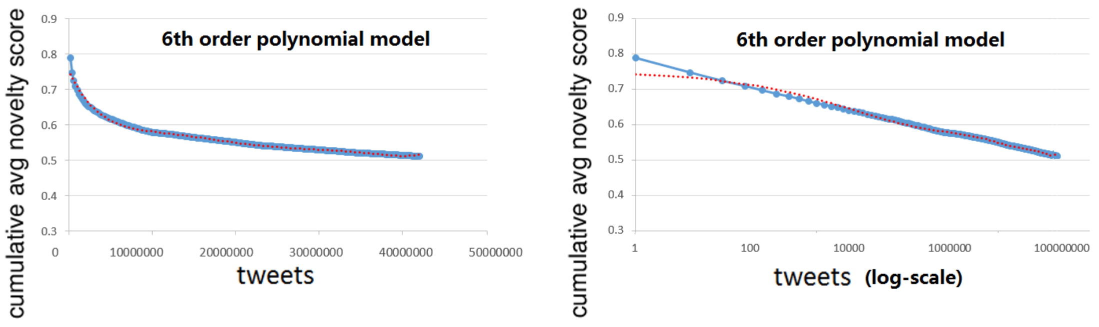


Figure 7.12: Illustration of novelty scores over time in linear and log-scale. The bold blue curve indicates the observed cumulative average novelty scores on Cross-Twitter. The red dotted line resembles the 6^{th} order polynomial model of the cumulative average novelty score from 52 million random tweets.

The optimal coefficient determinant for all three polynomial models and the training set (52 million random tweets), as well as the observed novelty scores from the test set (Cross-Twitter 52mio), are shown in Table 7.2. The table shows a lower proportional variance between the observed data of the training set and approximation from higher order polynomial functions, in comparison with lower order polynomial functions. This is expected as more powerful functions are better suited to closely resemble observed data. Interestingly, when applying the polynomial models to the test set, we measure lower coefficient determinants for 5^{th} order function ($R^2 = 0.9741$) than for 6^{th} order functions ($R^2 = 0.9733$). We conclude that 6^{th} order polynomial functions over-fit to the training set and don't generalise well to the test set.

Exponential Model

The exponential functions in Equation 7.2 is less powerful than the previously explored polynomial function. However, Figure 7.13 reveals that modelling the expected novelty decay based on exponential functions, allows capturing the novelty decay patterns observed on the test set.

$$EN(t) = \gamma * t^{\delta} \tag{7.2}$$

As before, we optimize the model parameters, γ and δ, on the training set consisting of 52 million random tweets. Table 7.2 show that the coefficient determinant of the exponential model ($R^2 = 0.9815$) is 1% higher on the test set than the best performing polynomial model ($R^2 = 0.9741$). This confirms that exponential model is superior to polynomial functions when modelling the expected novelty score decay with respect to a particular time t within the stream.

Logarithmic Model

Additionally, we model the cumulative average novelty scores using a logarithmic decay model, as seen in Equation 7.3.

$$EN(t) = \gamma * (-)ln(t) + \delta \tag{7.3}$$

The parameter γ denotes the slope, and δ is the intercept, which are optimized using the test set. We denote by t the time-stamp of a particular document position within the stream. The model, based on the inverted natural logarithm, highly correlates with the observed score decay, as seen in Figure 7.14 and Table 7.2. The coefficient determinant with the test set results in $R^2 = 0.9851$, which is the highest of all explored models.

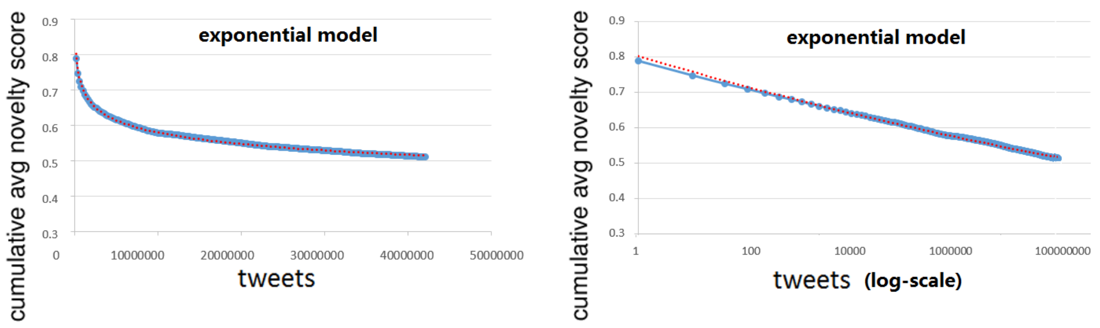


Figure 7.13: Illustration of novelty scores over time in linear and log-scale. The bold blue curve indicates the observed cumulative average novelty scores on Cross-Twitter. The red dotted line resembles the exponential model of the cumulative average novelty score from 52 million random tweets.

High coefficient values indicate low proportional variance between approximated and observed cumulative average novelty scores. Table 7.2 also shows that the two simpler decay models, based on the inverse of the exponent and logarithm, generalize better to the test set than higher order polynomial models.

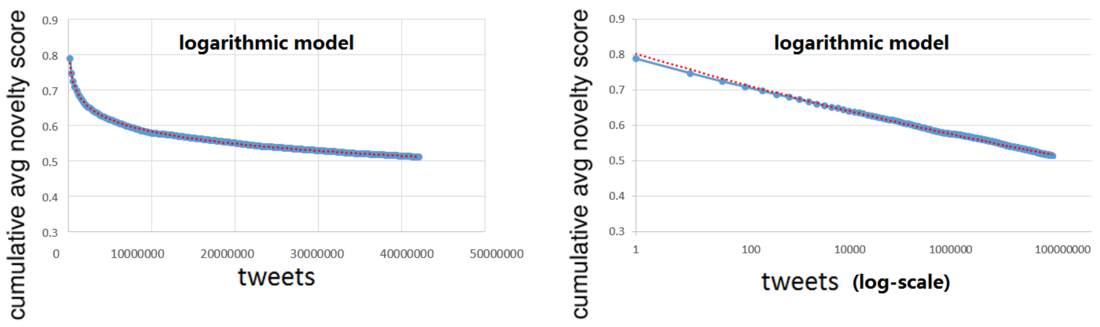


Figure 7.14: Illustration of novelty scores over time in linear and log-scale. The bold blue curve indicates the observed cumulative average novelty scores on Cross-Twitter. The red dotted line resembles the inverted logarithmic model of the cumulative average novelty score from 52 million random tweets.

7.5.2 Counteracting Novelty Decay

In the previous section, we showed that deletion is necessary to avoid novelty decay over time. We also derived mathematical decay models that allow approximating the expected cumulative average novelty decay with respect to a particular time in the stream. Our experiments revealed that of all explored model, a simple logarithmic

Model	coefficient determinant training set	coefficient determinant test set
4^{th} polynomial	0.975	0.9639
5^{th} polynomial	0.9861	0.9741
6^{th} polynomial	0.991	0.9733
exponential	0.9969	0.9815
logarithmic	**0.9987**	**0.9851**

Table 7.2: optimal coefficient determinant of models on training set (52 million random tweets) and coefficient determinant resulting from approximating novelty decay on test set (Cross-Twitter)

decay model approximates the observed decay best. We now apply the logarithmic decay model, as seen in Equation 7.3, to counteract the cumulative average novelty score decay. Counteracting score decay is only viable for kterm hashing based FSD, which does not require document deletion to scale to high stream volumes. In the following experiment, we also apply the decay model to the scores of UMass and LSH-FSD to outline that it successfully counteracts novelty decay in comparison based FSD systems. We optimize the model parameters on the coefficient determinant for the two comparison based and kterm based FSD systems on the training set. The resulting model is then used to approximate the expected score decay for each system for every document in the stream, whereas the expected score is subtracted from the computed score. Figure 7.15 illustrates the average novelty score for the three systems after adapting the novelty score according to the decay models. The curves of all systems remain constant, independently from the number of documents processed. This also applies for the first few documents, where deletion strategies experience an initial drop. Interestingly, the curve for LSH-FSD shows a minor dent during the first 10,000 documents. We assume this results from building the decay model based on the cumulative average. During the first few documents, the cumulative average is unstable and bounces before flattening out. As a consequence, the expected decay has a higher variance with respect to the observed decay for the first few documents.

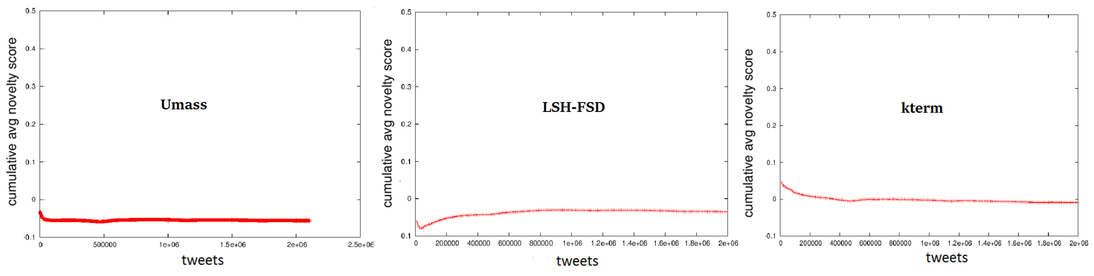


Figure 7.15: Illustration of the impact of adapting novelty scores (according to the decay model) on the average cumulative novelty score.

7.5.3 Impact of Adapting Novelty Scores on FSD Effectiveness

In the previous Section, 7.5.2, we showed that adjusting novelty score according to the proposed logarithmic decay model successfully prevents novelty scores from decaying over time. In the section, we explore the impact of score adjustment on FSD effectiveness and efficiency. Since FSD using kterm hashing is the only viable FSD system without a deletion strategy, we solely focus on kterm hashing in during the remainder of this chapter.

Our first experiment explores the impact of score adjustment through the logarithmic decay model on the detection accuracy of FSD using kterm hashing. We compare accuracy using DET plots instead, to illustrate the impact on the full range of possible detection settings. The DET plot for FSD using kterm hashing in Figure 7.17 compares kterm hashing without deletion, kterm hashing with random deletion and kterm hashing with decay model adjusted scores. Additionally, the DET plot also shows the 90% confidence corridor of normal kterm hashing by the two dotted blue lines. Any point outside the confidence corridor curves is statistically significantly different from normal FSD using kterm hashing.

DET plots show performance for all possible parameter settings. A particular parameter setting, consisting of the detection threshold that separates first stories from follow-ups, as well as the cost for miss and false alarms, depend on the application scenario. For example, an FSD system for earthquake detection comes with high costs of missing an earthquake. Consequently, this FSD system requires high detection performance for high recall scenarios. On the DET plot, recall is high when the miss probability is low. In a different scenario, investment banks might be interested accurately predicting certain new events, as false positives cause financial loss. This scenario requires a high precision setting. On the DET plot precision is high when the false alarm probability is low.

According to the DET in Figure 7.17, adjusting novelty scores does not significantly change accuracy for the high recall area of curves. This effect applies in particular where false alarm probabilities are $> 40\%$. High false alarm probabilities follow low detection thresholds, which cause follow-ups to be mistaken for first stories. Figure 7.16 illustrates that low threshold settings shrink area B. This explains why counteracting the decay by flattening the curve only marginally improves performance in the high recall region.

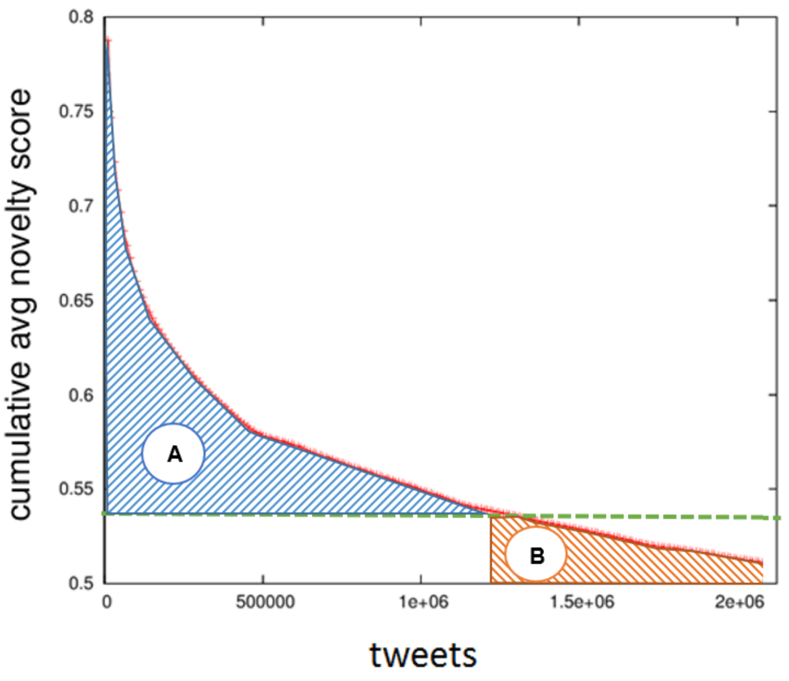


Figure 7.16: Illustration of the impact of novelty score decay on detection performance.

In contrast, the high precision area, where the false alarm rate is $< 15\%$, reveals benefits from adapting the novelty score according to the proposed model. The DET plot also illustrates that the difference between the model adjusted scores and the normal version is statistically significant. High precision scenarios shrink area A in Figure 7.16. Consequently, FSD systems are more likely to detect events early on and miss events that occur later in the stream. Flattening the decay curve increases the chance

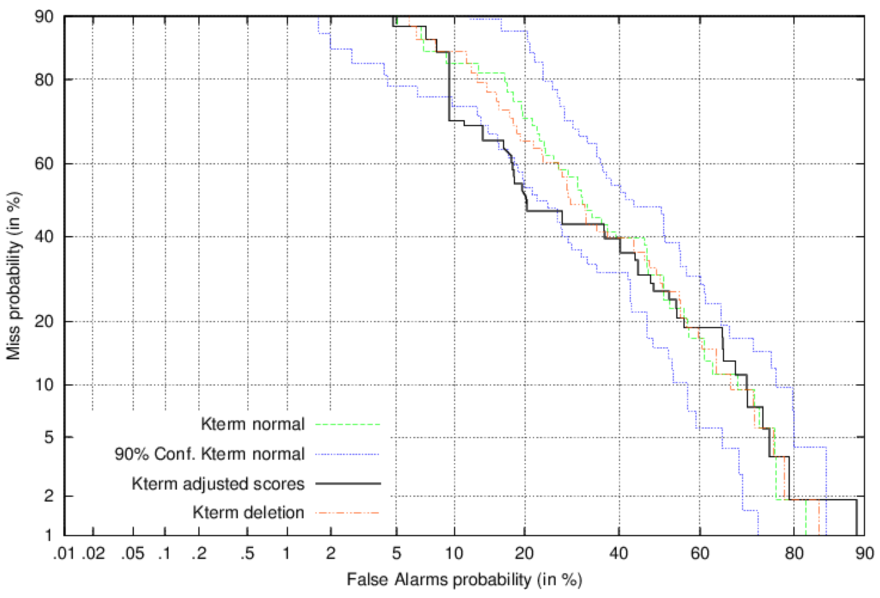


Figure 7.17: DET curve for kterm based FSD, comparing effectiveness of deletion through sliding window and score adjustment by model.

	Miss @ 10% False Alarm	**Dif.**
normal	83	-
deletion	85	+2%
adjusted scores	**69**	-17%

Table 7.3: Impact of counteracting novelty decay on the effectiveness of kterm hashing based FSD

of successfully detecting later events without increasing false-alarm probabilities, as seen in the DET curve.

We point out the improvement for a high precision setting by highlighting the performance at Miss Probabilities @ 10% False Alarm. Table 7.3 shows that the adjusting novelty scores according to the decay model, performs significantly better and decreases the miss probability by 17% over the normal set-up.

Document Deletion

In the previous section, we discovered that limiting the load factor of the memory successfully stops average novelty scores from decaying. Although successful on stopping novelty score decay, we cannot measure a significant positive impact on detection performance.

algorithm	**without deletion**	**random deletion**	**dif.**	**adjusted scores**	**dif.**
kterm	7,254 sec	13,669 sec	+90%	7,254 sec	0

Table 7.4: Impact of countermeasures on efficiency (measured by runtime in sec) for UMass, LSH-FSD and kterm hashing based FSD

7.5.4 Impact of Adapting Novelty Scores on FSD Efficiency

The following section explores the impact of score adjustment and deletion on FSD efficiency, measured by the runtime in seconds.Table 7.4 shows the runtime in seconds for FSD using kterm hashing on Cross-Twitter 52mio. According to the table, adapting the threshold based on the mathematical decay model does not influence the runtime of the algorithms. This is expected as the computational effort is only increased by a single scalar multiplication for each document processed. Consequently, adjusting novelty scores according to the proposed decay model does not influence the runtime of FSD algorithms.

In contrast, we measure a highly increased runtime when applying a deletion strategy to FSD using kterm hashing. Table 7.4 reveals that efficiency drops in half when deletion is applied. This effect is the same for all deletion strategies. Kterm hashing is a memory-based novelty estimation algorithm. As a result, novelty computation for kterm does not depend on the number of documents processed, but on the number of kterms hashed onto its memory. Deletion requires that for each new kterm, an old kterm has to fall out of scope. In order to zero out a kterm, it needs to be hashed again, to identify its corresponding cells in the memory. The deletion step doubles the number of hashing operations per document, which cuts efficiency in half.

Table 7.4 shows that FSD using kterm hashing processes Cross-Twitter in 7,252 seconds ($\sim$ 2 hours), which corresponds to a throughput of 7,170 tweets per second on a single core. To put this in perspective, LSH-FSD - the currently fastest FSD system that reaches accuracy comparable to exhaustive evaluation, requires 172,801 seconds ($\sim$ 48 hours) to process Cross-Twitter 52mio. LSH-FSD reaches a throughput of 300 tweets per second on a single core.

7.6 Conclusion

In the section, we studied the behaviour of novelty scores for comparison-based and non-comparison-based FSD systems without deletion strategies over time and revealed that they decrease over time. We also showed that the novelty score decay with the increase in space saturation. Deletion strategies place an upper limit and the space saturation, which prevents novelty score decay. Consequently, our experiments showed that deletion strategies not only speed-up the computation and retain operation with constant time and space but also ensure retaining a constant level of accuracy over time.

In this chapter, we also showed that the novelty score decay can be effectively modelled using a simple inverted logarithmic decay model. By calculating the expected decay with respect to a particular time in the stream, we successfully counteract the novelty decay by adopting document novelty scores accordingly. Our experiments studied the impact of score adjustment on FSD effectiveness and showed significantly increased effectiveness for high precision detection scenarios. Since score adjustment only adds a single scalar multiplication per document, the efficiency of FSD using kterm hashing remains unchanged.

Chapter 8

Further Applications of Novelty Detection

8.1 Motivation

Chapter 4 introduced a new method family for novelty computation called “memory-based novelty detection”. Memory-based algorithms extract concepts from a document and make them persistent in a single point of comparison, the memory. We then went on to introduce kterm hashing, the first member of the memory-based algorithm family. Kterm hashing computes a document’s novelty by modelling its concepts using kterms - non-empty sets of up to *k* distinct terms found in the document.

To demonstrate the capabilities of kterm hashing, we applied it to FSD, where new documents are compared to all previously seen documents. Kterm hashing improved the throughput of FSD systems by several orders of magnitude compared with state-of-the-art comparison based methods. In a nutshell, the challenge of FSD resides with comparing new documents to an ever growing set of old documents. Kterm hashing solves this challenge by combining all previous documents in a single representation, the memory, which provides a single point of comparison. Shifting the time complexity from the number of previous documents to the length of documents results in the fastest ever reported FSD throughput on a single core.

The following chapter is dedicated to demonstrate that memory-based algorithms, like kterm hashing, are capable of enriching other research tasks, beyond FSD. In particular, we build the first Rumour Detection system that detects rumours in realtime (zero lag). Rumour Detection on social media is a new trend that recently evolved into a popular research field. As is usual, we detect rumours using a detection model obtained from a Support Vector Machine (Chang et al., 2011), for which we introduce a new category of features, based on novelty.

Novelty features determine whether a tweet's information can be confirmed by a trusted resource. This resembles the NLP task of Information Entailment. Note that information entailment vastly differs from novelty detection for FSD. In our previous application of kterm hashing (FSD), novelty was defined by a document's similarity with respect to a set of previously encountered documents. This results in document-level novelty computation. By contrast, Information Entailment determines, how much of a document's information is entailed by another document. This requires sub-sentence-level comparisons.

Our experiments show that features build from memory-based algorithms significantly improve the accuracy when detecting rumour with zero lag. We also demonstrate how memory-based algorithms, like term hashing, are applicable to Mandarin by conducting our experiments for Rumour Detection on a Sina Weibo data set.

8.2 Rumour Detection

Social Media has evolved from friendship based networks to become a major source for the consumption of news. On social media, news is decentralised as it provides everyone the means to efficiently report and spread information. In contrast to traditional news-wire, information on social media is spread without intensive investigation, fact and background checking. The combination of ease and fast pace of sharing information provides a fertile breeding ground for rumours, false- and disinformation. Social media users tend to share controversial information in order to verify it, while asking about for the opinions of their followers (Zhao et al., 2015). This further amplifies the pace of a rumour's spread and reach. Rumours and deliberate disinformation have already caused panic and influenced public opinion.

The cases in Germany and Austria[1] in 2016, show how misleading and false information about crimes committed by refugees negatively influenced the opinion of citizens.

[1]http://hoaxmap.org/ on 15.02.2016

Rumour Detecting is the task of automatically determining whether a particular document is a rumour. The Cambridge dictionary defines a rumour as information of doubtful or unconfirmed truth. The original motivation of Rumour Detection was to prevent rumours from spreading and causing harm by refuting wrong factual claims. Unfortunately, the most potent features rely on a rumour's propagation patterns as well as the responses of other users. This is problematic as Rumour Detection systems operate retrospective and only detect rumours once they have spread. Consequently, automated rumour detection lags behind social media users in detecting and refuting rumours. By then, rumours have already spread and caused harm.

This highlights the importance and necessity of recognizing rumours as early as possible - preferably instantaneously.

We set out to detect rumour right after their publication. Note that this task is more challenging than traditional Rumour Detection, as future information about a tweet's distribution or others reaction is not available at a tweet's publication.

8.3 Approaches to Rumour Detection

Before rumour detection, scientists already studied the related problem of information credibility evaluation (Castillo et al., 2011; Richardson et al., 2003). Rumour detection on social media evolved into a popular research field that also relies on credibility assessment of messages and their sources. The most successful methods proposed focus on classification harnessing lexical, user-centric, propagation-based (Wu et al., 2015) and cluster-based (Cai et al., 2014; Liu et al., 2015; Zhao et al., 2015) features.

Many of these content based features originate from a study by Castillo et al. (2011), which pioneered in engineering features for credibility assessment on Twitter (Liu et al., 2015). They observed a significant correlation between the trustworthiness of a tweet with content-based characteristics including the presence and number of hashtags, punctuation characters and sentiment polarity. When assessing a tweet's credibility, they also consider the source of its information by constructing features based on URLs. Additional features for the credibility assessment rely on the Twitter-specific statistics of users, like their activeness and follow counts. One of the most successful feature categories is based on social graphs, assessing a message's re-tweet frequency and propagation patterns. Mendoza et al. (2010) found that the topology of a distrustful tweet's propagation pattern differs from those of news and normal tweets. These findings along with the fact that rumours tend to be more likely questioned by other users paved the way for future research examining propagation graphs and clustering methods (Cai et al., 2014; Zhao et al., 2015). The majority of current research focuses on improving the accuracy of classifiers through new features based on clustering (Cai et al., 2014; Zhao et al., 2015), sentiment analysis (Qazvinian et al., 2011; Wu et al., 2015) as well as propagation graphs (Kwon, et al., 2013; Wang et al., 2015).

The motivation for rumour detection lies in refuting wrong factual claims to prevent them from spreading and causing harm. Unfortunately, state-of-the-art systems operate in a retrospective manner, meaning they can only detect rumours long after they have spread. The most accurate systems rely on features based on propagation graphs and clustering techniques, which require rumours to spread before they can be detected.

To counteract the increasing lag between a tweet's publication and its detection as a rumour, researchers like Liu et al. (2015), Wu et al. (2015), Zhao et al. (2015) and Zhou et al. (2015) coined the term 'Early Rumour Detection'. Although Early Rumour Detection systems substantially decrease the lag between publication and detection, they still allow a delay of up to 24 hours. Within 1-day information can spread far in fast-moving social media services. However, their focus on latency aware Rumour Detection makes their approaches conceptually related to ours, with the difference that we require detection with zero lag. Zhao et al. (1015) found clustering tweets containing inquiry patterns as an indication of rumours and showed that a few messages questioning another one are sufficient for Rumour Detection. The low number of responses allows detecting rumours within 24 hours. The approach with the lowest reported latency also banks on the 'wisdom of the crowd' (Liu et al., 2015). In addition to traditional context and user based features, they rely on clustering tweets by their topicality to identify conflicting claims, which indicate increased the likelihood of rumours. Liu et al. (2015) claim to operate in real-time, but require a cluster of at least 5 messages to detect a rumour. Although the computation of their features is efficient, the lag of their approach depends on the delay with which other users question a message.

8.4 Improving Rumour Detection through Novelty Detection

We approach Rumour Detection task as a classification problem, as is usual in the research community, and assesses a documents likelihood of being a rumour. Counter to traditional Rumour Detection, we limit detection window to the time of a message's publication, requiring operation in real-time with a single pass over the data.

Problem Definition

More formally, we denote by d_t the document that arrives from stream $S : \{d_0, d_1, ...d_n\}$ at time t. Upon arrival of document d_t we derive its corresponding feature vector $f_{d,t}$. Given $f_{d,t}$ and the previously obtained weight vector w, we compute the rumour score $RS_{d,t} = w^T * f_{d,t}$. The rumour prediction is based on a fixed thresholding strategy with respect to threshold θ. We predict that message d_t is likely to be a rumour if its rumour score exceeds the detection threshold: $RS_{d,t} > \theta$. The optimal parameter setting for weight vector w and detection threshold θ are learned using a Support Vector Machine (Chang et al., 2011) on a training set to maximise prediction accuracy.

The most successful features for Rumour Detection rely on information that can only be computed retrospectively. This renders them close to useless when detecting rumours early on. We introduce two new categories of features, one based on novelty, the other on pseudo feedback. Both feature categories aim at improving detection accuracy early on when information is limited.

8.4.1 Approach Overview

Our approach to Rumour Detection is based on traditional content based features in addition to our novelty based features and pseudo feedback. Figure 8.1 illustrates a system diagram of the detection phase.

For each document (doc_n) arriving from the stream (Weibo stream), we compute its feature values. In total, we base our detection model on 57 features including features from our two new feature categories, novelty based features and pseudo feedback. We then linearly combine all feature values with their corresponding weights according to

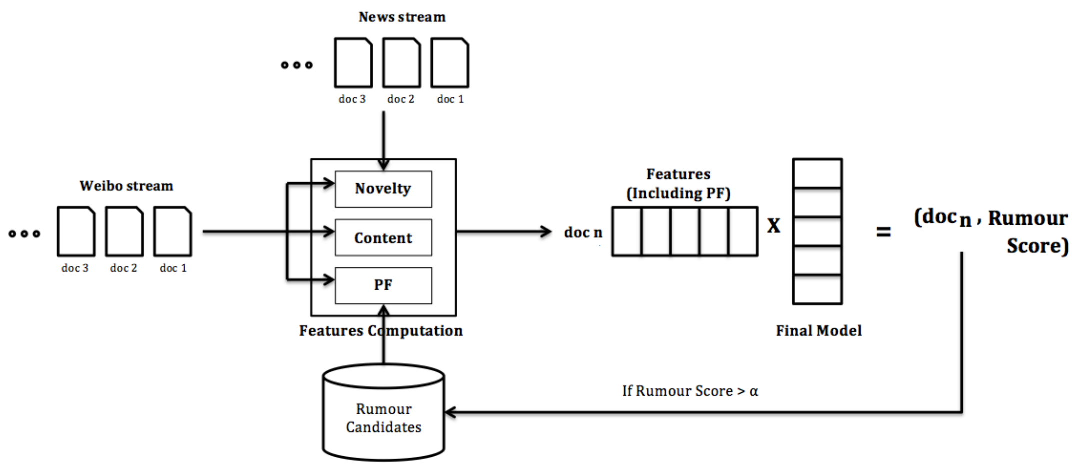


Figure 8.1: System diagram illustrating our approach to Rumour Detection using novelty based, content based and pseudo feedback features.

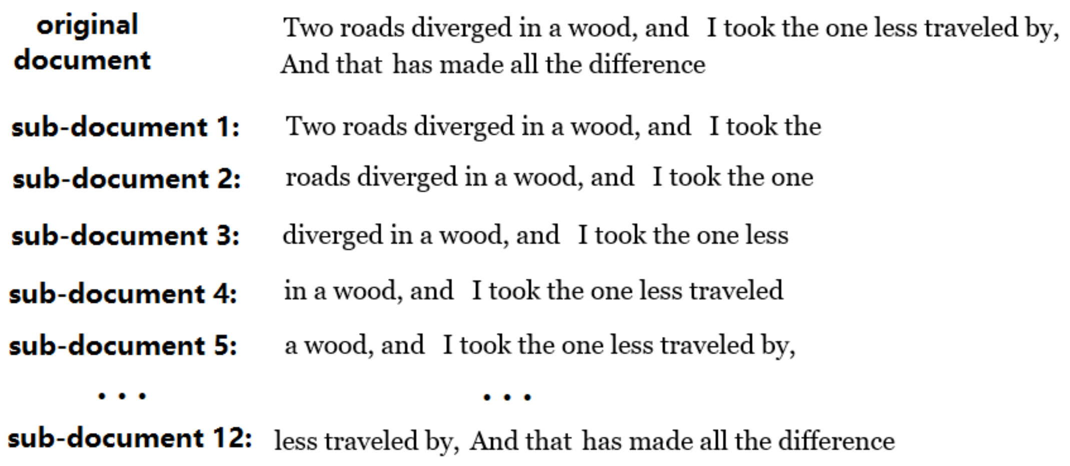
the detection model, forming the final rumour score. The detection model holds the feature weights learned by a Support Vector Machine on the training set. Our system emits a document-identifier and rumour score tuple. As for FSD, we apply a fixed thresholding strategy to separate rumours from non-rumours. Documents whose rumour score exceeds the detection threshold α are considered rumours candidates.

The following section introduces our two new feature categories and elaborates on the training phase.

8.4.2 Novelty based Features

We restrict Rumour Detection to decision making with zero lag. Traditional Rumour Detection accuracy is low when limited to decision making shortly after a document's publication. To increase instantaneous detection accuracy, we compensate for the absence of future information by consulting additional data sources. In particular, we make use of news-wire articles, which are considered to be of high credibility. When a message arrives from a social media stream, we build features based on its novelty with respect to the confirmed information in the trusted sources. In a nutshell, the presence of information unconfirmed by the official media is construed as an indication of being a rumour. This is reasonable, since Petrovic et al. (2013) found that in the majority of cases, news-wires lead social media for reporting news, except for sports events. Note that verifying whether a rumour candidate's information is confirmed closely resembles the definition of what a rumour is.

High-volume streams demand highly efficient feature computation. This applies in particular to novelty based features since they can be computationally expensive. We explore two two approaches for novelty computation: 1) the traditional comparison based method, based on the on vector proximity of the cosine distance 2)kterm hashing, our memory-based method.



original document Two roads diverged in a wood, and I took the one less traveled by, And that has made all the difference

sub-document 1: Two roads diverged in a wood, and I took the

sub-document 2: roads diverged in a wood, and I took the one

sub-document 3: diverged in a wood, and I took the one less

sub-document 4: in a wood, and I took the one less traveled

sub-document 5: a wood, and I took the one less traveled by,

• • • • • •

sub-document 12: less traveled by, And that has made all the difference

Figure 8.2: Example of a term level sliding window of length 10, separating the original document into 12 sub-documents.

Comparison Based

The first method applies traditional vector proximity based on 1 minus the cosine similarity between two documents:

$$Novelty(d_1, d_2) = 1 - \frac{\sum_{i=1}^{n} d_{1,i} * d_{2,i}}{\sqrt{\sum_{i=1}^{n} d_{1,i}^2} * \sqrt{\sum_{i=1}^{n} d_{2,i}^2}} \tag{8.1}$$

The cosine similarity alone yields poor performance due to the length discrepancy between news-wire articles and social media messages (Wurzer et al., 2015). To make vector proximity applicable we slide a term-level based window, whose length resembles the average social media message length, through each of the news articles, as seen in Figure 8.2. This results in sub-documents of comparable length as social media messages. Novelty is computed using term weighted tf-idf cosine similarity between the social media message and all news sub-documents, using Equation 8.1. The minimum novelty score between a message and all sub-documents approximates the degree confirmation through the trusted resources.

memory-based

The second approach for novelty based features applies kterm hashing. Instead of measuring the similarity between documents, we construct a single memory spanning all available trusted resources, as seen in Equation 8.2.

$$\forall (d \in TR) : M \leftarrow M \cup \{t : t \subset d\} \tag{8.2}$$

By M, we denote the memory, that holds the information (kterms t) of all documents d in the set of trusted resources TR. As for FSD, the degree of a document's novelty is computed by the fraction of unseen kterms with respect to the memory. In contrast to its application in FSD, kterm hashing is now used to assess, whether the information of a short message (tweet) is entailed by a set of long documents (news articles).

When we introduced kterm hashing in Chapter 4, we distinguished kterm by their cardinality, the number of compounded terms. Our experiments showed that novelty detection improves when placing individual weights based on kterm cardinality. Although FSD data sets have high numbers of documents, the number of annotated topic is insufficient to learn optimal weight settings. The labelling procedure for Rumour Detection requires less manual effort, which allowing us to split our data set into a test and training set. The training data set allows us to learn optimum weight for each level of kterm cardinality. Instead of a single novelty based feature we compute a novelty score for each cardinality level, using Equation 8.3.

$$Nl(d_n) = \sum_{T \in \{T : T \subset c_n, |T| = l\}} \binom{|d_n|}{|T|}^{-1} \begin{Bmatrix} 1 : T \notin M \\ 0 : T \in M \end{Bmatrix} \tag{8.3}$$

This leaves us with a separate feature (novelty score) for each level of cardinality.

8.4.3 Applying Kterm Hashing to Mandarin

Our data set for Rumour Detection uses messages from Sina Weibo, a Chinese social media service for the public distribution of micro-blogs, comparable to Twitter. The following section describes how kterm hashing is applied to text written in Mandarin.

Term Segmentation

Kterms are sets of terms from a document. Western languages including English provide term boundaries, which are necessary to construct kterms. Mandarin is a languages that does not provide word boundaries[2], as seen in Figure 8.3. Text processing in Mandarin requires an initial segmentation process to break up the continues text into terms (Levy et al., 2005). In a first attempt, we segment on each character forming 1-grams. The result does not always represent actual terms, which can consist of a single or more characters. Splitting Mandarin strings into fixed length n-grams represents the simplest and fastest segmentation method (Foo et al., 2004). As an alternative to uni-grams, we also segment characters into 2-grams and 3-gram, which Foo et al. (2004) found superior to uni-grams for information retrieval tasks. When segmenting 2-grams and 3-grams, we acknowledge existing blanks, as well as numbers and sentence characters as natural segmentation points, as seen in Figure 8.3.

In addition to simple rule based segmentation we also apply the Stanford Word Segmenter (Tseng et al., 2005), which was designed to segment Mandarin texts to aid IR and NLP tasks. The Segmenter successfully participated in the 2005 Sighan Bakeoff[3] and applies statistical sequence modelling using conditional random fields in conjunction with character identity features, morphological features and character reduplication features.

Figure 8.3 shows that the Word Segmenter successfully breaks the Mandarin character sequence into real words. We explore the impact of word segmentation on the accuracy of novelty based features in Section 8.5.

Kterm on longer documents

Twitter applies a 140 character limit, which corresponds an average tweet length of 10 terms. Sina Weibo also restricts messages to 140 characters. However, in Mandarin a single character can represent an entire term, resulting in an average document length of 90 terms. Although the character limits of both micro-blog services matches, messages on Sina Weibo contain on average 9 times more terms than tweets. The number of distinct terms in a message determines the number of kterms in its content. Equation

[2]https://en.wikipedia.org/wiki/Category:Writing_systems_without_word_boundaries

[3]http://www.sighan.org/bakeoff2005/

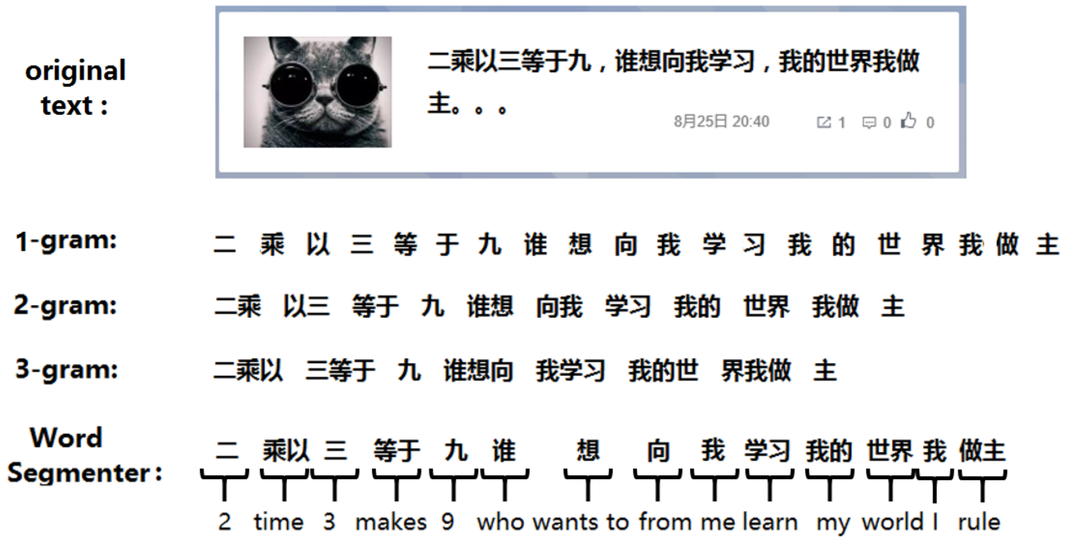

original text :
二乘以三等于九，谁想向我学习，我的世界我做主。。。
8月25日 20:40
1
0
0
1-gram:
二 乘 以 三 等 于 九 谁 想 向 我 学 习 我 的 世 界 我 做 主
2-gram:
二乘 以三 等于 九 谁想 向我 学习 我的 世界 我做 主
3-gram:
二乘以 三等于 九 谁想向 我学习 我的世 界我做 主
Word Segmenter :
二 乘以 三 等于 九 谁 想 向 我 学习 我的 世界 我 做主
2 time 3 makes 9 who wants to from me learn my world I rule

Figure 8.3: Example message from Sina Weibo, a social media service in China comparable to Twitter, showing that texts in Mandarin come without term boundaries; The figure also shows term segmentation using 1-3 grams and the Stanford Text Segmenter.

4.4 in Chapter 4 illustrates that the content size arises from the binomial coefficient between the document size and the kterm cardinality. The content of an average tweet contains $\binom{10}{3} = 120$ kterms. By contrast, the content of an average message from Sina Weibo contains $\binom{90}{3} = 117,480$ kterms, which is 979 times more than a tweet.

Each kterm requires a novelty check with respect to the memory, which includes hashing the string on the bit array and a subsequent check whether the corresponding bit is set. To minimize the impact of 3 order of magnitude more kterms on the runtime, we limit the kterms by only considering the top k. Limiting the number of features (terms) of a document is a widely applied method in IR and TDT to reduce computational effort and increase algorithm efficiency (Yang et al., 1998). In our experiments in Section 8.5, we explore the impact of limiting documents to the top 10 $tf.idf$ weighted terms and subsequently from kterms from them, for Rumour Detection. We empirically determined to limit the number of features to 10, which also resembles the average number of terms in a tweet.

8.4.4 Pseudo Feedback

In addition to novelty based features, we introduce another category of features - dubbed Pseudo-Feedback (PF) - to boost detection performance. The feature is conceptually related to pseudo-relevance feedback found in retrieval and ranking tasks in IR. Pseudo-relevance feedback is applied to retrieval task to boost the effectiveness of document ranking algorithms. Using a query, an initial retrieval step identifies the k documents that most similar to the query. Pseudo-relevance feedback considers these documents as relevant and expands the query by terms from these documents. A consecutive second retrieval step, using the expanded query, determines the final document ranking.

The concept of our pseudo feed feature builds upon the idea that documents, which reveal similar characteristics in term space as previously detected rumours, are also likely to be a rumour. During detection, feedback about which of the previous documents describes a rumour is not available. Therefore, we rely on 'pseudo' feedback and consider all documents whose rumour score exceeds threshold α as true rumours. The PF feature describes the maximum similarity in term space between a new document and those documents previously considered as rumours. Similarities are measured by vector proximity using cosine similarity (Equation 2.4). PF passes evidence of being a rumour to future documents that are similar to previous documents considered as rumours. Note that this allows harnessing information from repeated signals without operating retrospectively.

Training Pseudo Feedback Features
The training routine of PS differs from the standard procedure because the computation of the PF feature requires two training rounds. Figure 8.4 illustrates the initial training round, during which an SVM is used to compute the weights for all features in the training set, except the PF features. This provides a model for all but the PF features. We then process the training set, as seen in Figure 8.5, to compute rumour scores based on the model obtained from our initial training round. Document's whose rumour score exceeds θ are considered as rumour candidates. Figure 8.6 shows the second training round that learns the PF feature weights by measuring the minimum distance in term space between the current document vector and those rumour candidates that pre-date it. To speed up computation, we only compare against the k most

recent rumour candidates. Once we obtained the weight for the PF feature we combine it with the weight vector computed in the initial training round to form the final model, as seen in Figure 8.7.

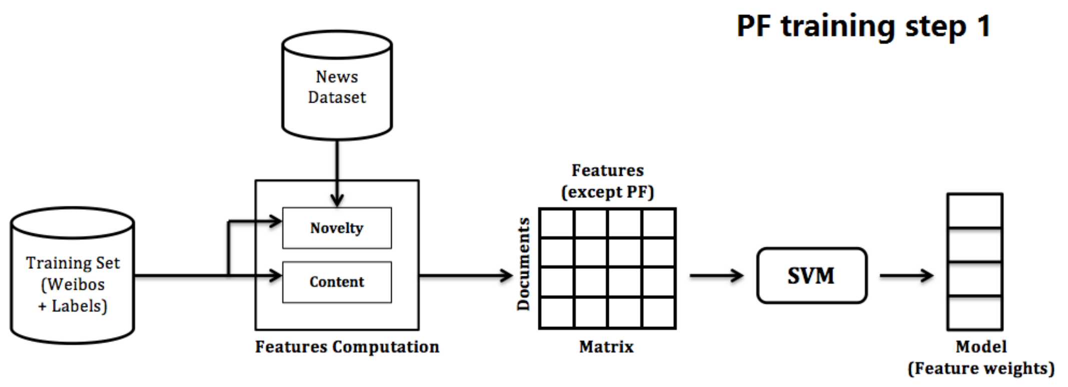


Figure 8.4: Initial training round to learn the weights of all but the PF feature.

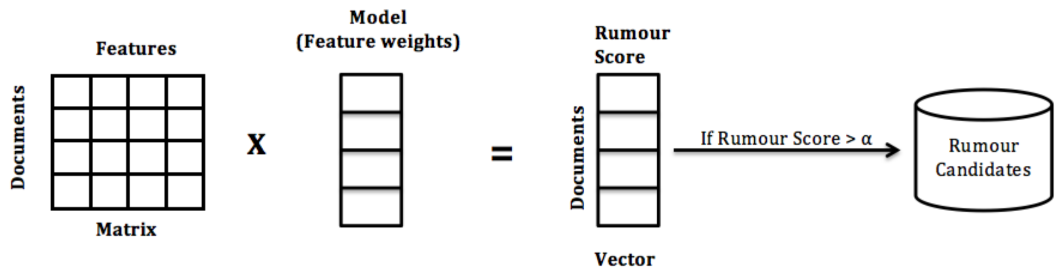


Figure 8.5: Weights of initial training round are used to detect rumour candidates that are subsequently used to train pf feature.

8.4.5 Context Based Features

In addition to our novelty based features and pseudo feedback, we also apply a range of context based features. The focus lies on features that can be computed at the time of a message's publication. In particular, we compute apply a range of 51 context based features, that can be ordered into 7 feature categories, as seen in Table 8.1. Most of these 51 features overlap with previous studies (Castillo et al., 2011; Liu et al., 2015; Qazvinian et al., 2011; Yang et al., 2012; Zhao et al., 2015). This includes features based on the presence or number of URLs, hash-tags and user-names, as well as the

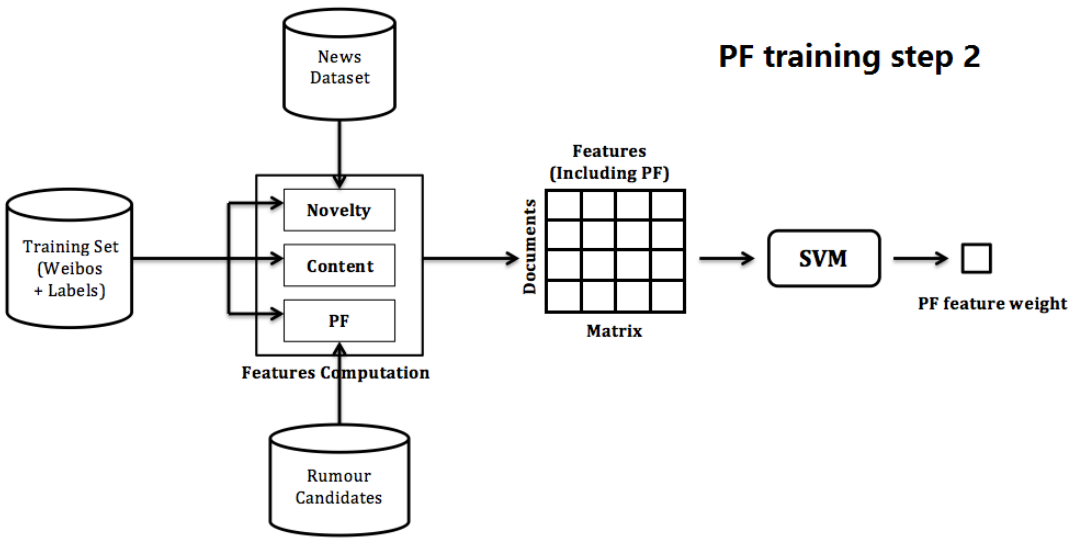


Figure 8.6: Second training round to learn PF feature weight using the rumour candidate set.

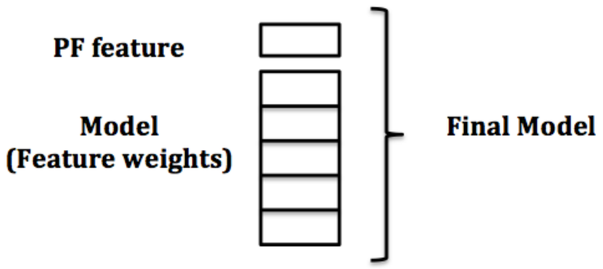


Figure 8.7: Final model consists of the feature weights from initial training round and PF weight from the second training round.

number of certain POS tags and punctuation characters. We also distinguish between 8 different categories of sentiment and emotions. We don't consider user based features because they are specific to social media services and not universally applicable.

Category	Description
Punctuation	categorical feat. based on the number of !?.,
POS	categorical feat. based on the number of verbs, nouns, adjectives, quantity and time words
Sentiment	categorical feat. based on the number of strong/weak negative/positive words
Emotion	degree of positive/negative/sad/anxious/surprised emotion
Social Media	categorical feat. based on the number of hash-tags and user-names
Length	categorical feat. based on the number of unique words
URLs	categorical feat. based on the number of URLs, pictures
Novelty	novelty score based kterm length 1-3 for all and key-words
Pseudo Feedback	distance to the closest previous rumour

Table 8.1: Description of features

8.5 Experiments

The previous sections introduced two new categories of features for rumour detection. Now we test their performance and impact on detection effectiveness and efficiency. In a streaming setting, documents arrive on a continual basis one at a time. We require our features to compute a rumour-score instantaneously for each document in a single pass over the data. Messages with high rumour scores are considered likely being rumours. The classification decision is based on an optimal thresholding strategy based on the training set.

8.5.1 Evaluation Metrics

We report accuracy to evaluate effectiveness, as is usual in the literature (Zhou et al., 2015). Additionally, we use the standard TDT evaluation procedure (Allan et al., 2000; NIST, 2008) with the official TDT3 evaluation scripts (NIST, 2008) using standard settings, which we introduced in Chapter 2. This procedure evaluates detection tasks using Detection Error Trade-off (DET) curves, which show the trade-off between miss and false alarm probability. By visualizing the full range of thresholds, DET plots provide a more comprehensive illustration of effectiveness than single value metrics (Allan et al., 2000). We also evaluate the efficiency of computing the proposed features, measured by the throughput per second, when applied to a high number of messages.

8.5.2 Data Set

Rumour detection on social media is a novel research field without official data sets. Since licence agreements forbid redistribution of data, no data sets from previous publications are available. Consequently, we followed researchers like Liu et al. (2015) and Yang et al. (2012) and created our own dataset. Preprocessing is limited to splitting on punctuation without removing anything. In contrast to FSD we do not remove usernames, hashtags and punctuation characters because we build features based on their presence or absence.

Trusted Resources

We randomly collected 200 news articles about broad topics commonly reported by news-wires over our target time period. These range from news about celebrities and disasters to financial and political affairs as seen in table 8.8. Since we operate on Chinese social media, we gathered news articles from Xinhua News Agency[4], the leading news-wire in China. To ensure a fair evaluation, we collected the news articles before judging rumours, not knowing which rumours we would find later on. We also only consider news articles published before the timestamps of the social media messages.

Rumours

We chose Sina Weibo, a Chinese social media service with more than 200 million active users[5] as our social media stream. Micro-blogs from Sina Weibo are denoted as '*weibos*'. Weibos are comparable to tweets, with which they share a 140 character limit. As tweets, all weibos are publicly accessible and offer hashtags and directed messages using usernames.

Sina Weibo provides an official rumour debunking service, operated by trained human professionals. Following Yang et al. (2012) and Zhou et al. (2015), we use this service to obtain a high-quality set of 202 confirmed rumours.

Non-Rumours

[4]http://www.xinhuanet.com/

[5]http://www.bbc.co.uk/news/technology-35361157 as of 10.02.2013

We additionally gathered 202 non-rumours using the public Sina Weibo API[6]. Three human annotators judged these weibos based on unanimous decision making to ensure that they don't contain rumours.

Since we operate in a streaming environment, all weibos are sorted based on their publication time-stamp. Table 8.3 shows a list of example for rumours found in our data set.

We ordered the rumours and non-rumours chronologically and divided them in half, forming a training and test set. We ensured that each of the sets consists of 50% rumours and non-rumours. This is important when effectiveness is measured by accuracy. All parameters and weights are learned and optimised using the training set. Performance is reported based on a single run on the test set.

8.5.3 Rumour Detection Effectiveness

To evaluate our new features for rumour detection, we compare them with two state-of-the-art early rumour detection baselines Liu et al. (2015) and Yang et al. (2012), which we re-implemented. We chose the algorithm by Yang et al. (2012), dubbed Yang, because they proposed a feature set for early detection tailored to Sina Weibo and were used as a state-of-the-art baseline before by Liu et al. (2015). The algorithm by Liu et al. (2015), dubbed Liu, is said to operate in real-time and outperformed Yang, when only considering features available on Twitter. Both apply various content based, user based, topic based and propagation based features and rely on an SVM classifier, which they found to perform best. The approaches advertise themselves as suitable for early or real-time detection and operate with the smallest latency across all published

[6]http://open.weibo.com/wiki/API

Algorithm	**Our App.**	**Liu**	**Yang**
Accuracy	**75%**	62.26%	60.21%
Difference	-	-17%	-20%

Table 8.2: Effectiveness in comparison with two state-of-the-art baselines for instant rumour detection using optimal thresholds

马航MH370最新消息，遭恐怖分子劫持过程曝光，点击查看 MH370 latest news, the process of being hijacked by terrorist has been revealed, click to view details
重大新闻，苹果公司获得三星10亿美元赔款，整整30 卡车硬币开到了苹果公司的总部！！！ Breaking news, Apple just got the 1 billion dollars for reparation from Samsung, Samsung drove 30 trucks of coins to the headquarters of Apple.
法对叙IS第一轮战术核打击定于16日上午10时进行。 France will start the first round nuclear attack against IS at 10:00 a.m. on the 16th

Table 8.3: Examples rumours with translation

Algorithm	**instant**	**12 hours**	**24 hours**
our App.	**75%** *	**75%**	75%
Liu	62.27%	74.29%	**78.4%**
Yang	60.21%	65.35%	75.82%

Table 8.4: Detection accuracy at different levels of delay; Asterisk indicates significance ($p < 0.05$)

methods. Yang performs early rumour detection and operates with 24 hours delay. Liu is claimed to perform in real-time, while requiring a cluster of 5 repeated messages to judge them for rumours. Note that although these algorithm are state-of-the-art for detecting rumours as quickly as possible, they still require a certain delay to reach their full potential.

Table 8.2 compares the performance of our approach with the two classifiers on the 101 rumours and 101 non-rumours of the test set, when detecting rumour instantly after their publication. The table reveals comparable accuracy for Yang and Liu at around 60%. Our observed performance of Yang matches those by Liu et al. (2015). Surprisingly, the algorithm Liu does not perform significantly better than Yang when applied to instantaneous rumour detection although they claimed to operate in real-time. Liu et al. (2015) report performance based on the first 5 messages which clearly outperforms Yang for early rumour detection. However, we find that when reducing the set from 5 to 1, their superiority is only marginal. In contrast, the combination

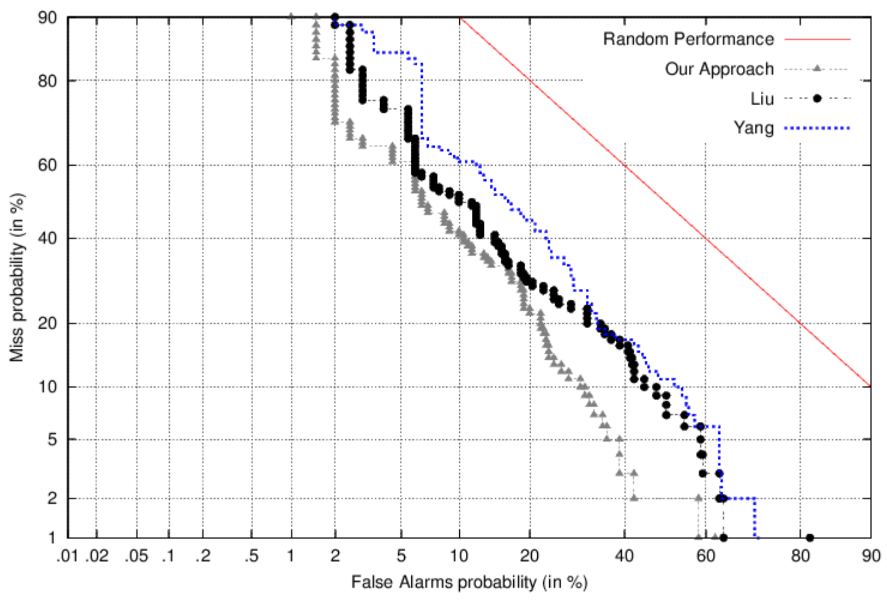


Figure 8.8: DET plot, revealing superior effectiveness of our approach for instant rumour detection for the full range of thresholds.

of novelty and pseudo-relevance based features performs significantly better (sign test with $p < 0.05$) than the baselines for instantaneous rumour detections. Novelty based features benefit from news articles as an external data source, which explains their superior performance when other information is limited. In particular for instantaneous rumour detection, where information can only be obtained from a single message, the use of external data proves to perform superior.

Note that accuracy is a single value metric describing performance at an optimal threshold. Figure 8.8 compares the effectiveness of the three algorithms for the full range of rumour scores for instantaneous detection. Different applications require a different balance between miss and false alarm. But the DET curve shows that Liu's method would be preferable over Yang for any application. Similarly, the plot reveals that our approach dominates both baselines throughout all threshold settings and for the high-recall region in particular.

When increasing the detection delay to 12 and 24 hours, all three algorithms reach comparable performance with no statistically significant difference, as seen in table 4. For our approach, none of the features are computed retrospectively, which explains why the performance does not change when increasing the detection delay. The additional time allows Liu and Yang to collect repeated signals, which improves their detection accuracy. After 24 hours Liu performs the highest due to its retrospectively computed features. Note that after 24 hours rumours might have already spread far through social networks and potentially caused harm.

Features:	**Accuracy**
all	75%
sentence char	69%
POS	71%
emotion	73%
extreme words	72%
sentiment	73%
PF	71%
novelty (kterm all + kterm top k + cosine subdoc)	**60%**

Table 8.5: **Features ablation**: impact on performance when removing feature groups. **Note**: lower accuracy means higher impact; POS: part of speech; PF: pseudo-feedback; subdoc: cosine similarity using sub-documents resulting from sliding window

Novelty based Features:	**Accuracy**
cosine	64%
cosine subdoc	69%
kterm (all)	70%
kterm (top k)	74%
kterm all + kterm top k + cosines subdoc	**75%**

Table 8.6: Impact of different kinds of novelty based features on Rumour Detection accuracy. **Note**: This table shows performance when applying all other features in addition to novelty based features.

8.5.4 Feature Analysis

We group our 57 features into 7 categories shown in Table 8.1 and analyse their contribution using feature ablation, as seen in Table 8.5. Feature ablation illustrates the importance of a feature by measuring performance, when removing it from the set of features. Consequently, lower accuracy indicates a higher importance of a features. The table shows that novelty based features were found to be dominant for instantaneous rumour detection ($p < 0.05$). 'Sentence char' features, which include punctuation, hashtags, user-symbols and URLs, contributed the most of the traditional features, followed by Part of Speech ('POS') and 'extreme word' features. Our experiments found 'sentiment' and 'emotion' based features to contribute the least. Since excluding them both results in a considerable drop of performance we conclude that they capture comparable information and therefore compensated for each other.

Novelty based Features

Novelty based features revealed the highest impact on detection performance. To further analyse the impact of different kinds of novelty based features we measure they individual contribution on detection accuracy. Table 8.6 show the performance of our Rumour Detection system when applying all other features in conjunction with a particular kind of novelty based feature. The table shows that dividing the trusted resources into sub-documents through a sliding window is beneficial, as it increases accuracy by 5% (absolute). As explained in Section 8.5.4, cosine similarity is suboptimal when determining vector proximity between imbalanced document lengths. Term weights in short documents, which many only containing a single term, are inflated by the cosine's length normalization and consequently distort novelty approximation. When striping all but the top 25 tf-idf weighted terms from the news sub-documents, the hit in performance can be reduced by 3 % (absolute).

Table 8.6 also shows that novelty based feature using kterm hashing result in higher accuracy than features based on vector proximity by cosine similarity. In particular kterms formed from the top keywords contribute the most. This is surprising, as we limited the number of term in section 7.2 to increase the efficiency of kterm hashing on longer documents. Instead of forming kterms from all terms we only the top $tf.idf$ weighted terms when constructing kterms. So far we distinguished kterms by their cardinality, which allows assigning weights based on the cardinality level. The increase in detection accuracy provides evidence that it is beneficial to also distinguishing kterm based on the weight of the terms they are formed of.

Novelty features based on kterm hashing construct a combined memory of all information presented to it. Pulling all information into a single representation bridges the gab between documents and allows finding information matches within documents. We hypothesize that this causes increased detection performance over cosine based novelty features.

Pseudo Feedbaack

Features ablation in Table 8.5 revealed that pseudo feedback (PF) increased detection performance by 5.3% (relative). PF builds upon the output of the other features. High performance of the other features results in higher positive impact of PF. We plan to further explore the behaviour of PF on other detection tasks in future studies.

Novelty based Features:	**Accuracy**
1-gram	63%
2-gram	62%
3-gram	59%
Stanford Word Segmenter	**70%**

Table 8.7: Impact of word segmentation for Mandarin on detection accuracy using kterm hashing on all terms

8.5.5 Impact of Term Segmentation on Detection Accuracy

According to Foo et al. (2004), assuming all terms to be n-grams provides the fastest method to segment texts in Mandarin. Table 8.7 reveals that n-gram based segmentation heavily reduces detection accuracy in comparison with a more sophisticated word segmenter. Even-tough we consider natural text breaks and sentence characters none of the n-grams performs comparable to the segmenter. Comparison of Table 8.7 and Table 8.1 shows that segmenting by 3-gram renders novelty based features useless. We conjecture that in Mandarin words that consist of 3 or more characters are less likely and therefore reduce detection accuracy. The combination of different levels of n-gram did not improve Rumour Detection accuracy. We conclude that kterm hashing requires accurate word boundaries to reach adequate performance.

8.5.6 Detecting Unpopular Rumours

Previous approaches to rumour detection rely on repeated signals to form propagation graphs or clustering methods. Beside causing a detection delay these methods are also blind to less popular rumours that don't go viral. In contrast, novelty based feature require only a single message enabling them to detect even the smallest rumours.

Topic Name	Rumour synopsis
Terror in Paris	fake news report of Chinese hostages and their cruel torture by terrorist disguised as Syria refugees
RUS plane crash	various different rumours blaming Turkey, US and Russia itself
Samsung vs Apple Patent law suit	rumors that Samsung paid Apple the fine with trucks of coins to show their dissatisfaction with the judgement
Flight MH370	rumours about the cause of the crash including abduction by terrorists, shot down by US military and CIA

Table 8.8: Excerpt of topics with synopsis of corresponding rumours

Examples for such small rumours are shown in table 8.3.

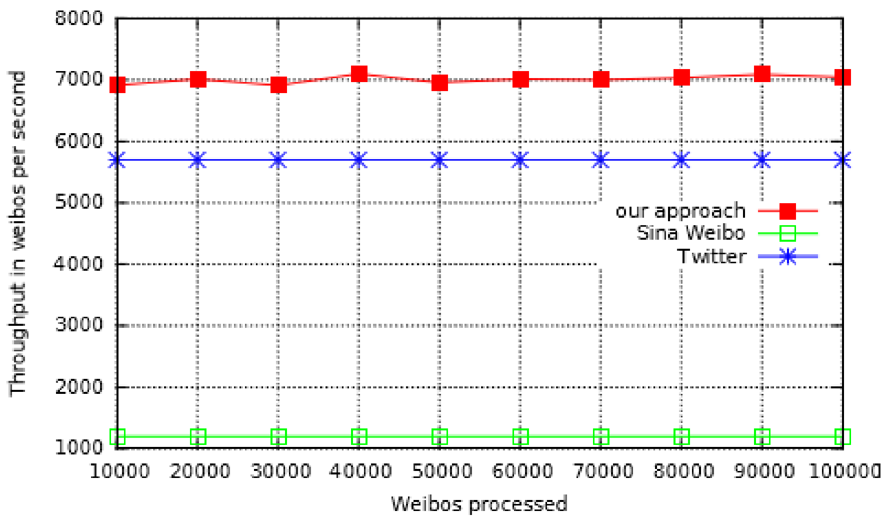


Figure 8.9: Throughput of our approach per second in comparison to the average Twitter (Firehose) and Sina Weibo stream.

8.5.7 Efficiency and Scalability

To demonstrate the high efficiency of computing novelty and pseudo feedback features, we implement a rumour detection system and measure its throughput when applied to 100k weibos. We implement our system in C and run it using a single core on a 2.2GHz Intel Core i7-4702HQ. We measure the throughput on an idle machine and average the observed performance over 5 runs. Figure 8.9 presents performance when processing more and more weibos. The average throughput of our system is around 7,000 weibos per second, which clearly exceeds the average volume of the full Twitter[7] (5,700 tweets/sec.) and Sina Weibo[8] (1,200 weibos/sec.) stream. Since the number of news articles is relatively small, we find no difference in terms of efficiency between computing novelty features based on kterm hashing and vector similarity. Figure 8.9 also illustrates that our proposed features can be computed in constant time with respect to the number of messages processed. This is crucial to keep operation in a true streaming environment feasible. Approaches, whose runtime depend on the number of documents processed become progressively slower, which is inapplicable when operating on data streams. Our experiments show that the proposed features perform effectively and their efficiency allows them to detect rumours instantly after their publication.

8.6 Conclusion

In this section we showed that novelty detection using kterm hashing is applicable beyond FSD. In particular, we improved the accuracy of instant Rumour Detection by creating two new feature categories, novelty based features and pseudo feedback. Rumour Detection currently only operates retrospectively, and lag behind social media users for identifying messages as rumours. Our features are designed to improve deteciton accuracy at the time of a messages publication. Novelty based features judge whether the information of a message is confirmed by trusted resources. We compared novelty based features using cosine similarity and kterm hashing and found the latter superior when detecting rumours. Our second new feature category, called pseudo feedback, allows harnessing repeated signals without the need of retrospective operation and using a single pass over the data. In a nutshell, pseudo feedback considers documents that are close in term space to previous rumours, likely to also be a rumour.

Our experiments showed that our approach significantly outperforms the state-of-the-art systems with the lowest latency for instant Rumour Detection. Our feature analysis revealed that in particular novelty based feature increase detection accuracy for instant detection, when other information is limited.

While applying Rumour Detection to a Sina Weibo data set, we showed how kterm hashing can operate on text written in Mandarin. Since messages from Sina Weibo

[7]about.twitter.com/company (last updated: 31.3.2015)
[8]http://www.techweb.com.cn/internet/2012-01-06/1139327.shtml

are 9 times longer than tweets, we reduced them to the top k terms to limit the impact on efficiency. Interestingly, our experiments showed that limiting the number of terms before forming kterms also increases Rumour Detection accuracy. We conclude that it is beneficial to distinguish kterms based on their cardinality and the weight of the terms they are formed of.

Chapter 9

Conclusion

This thesis presented a new algorithm family called memory-based novelty detection and applied it to event detection in unbounded streams of unstructured social media texts. There are three main contributions in this thesis: First, we introduced kterm hashing, a memory-based novelty computation method, whose throughput exceeds the currently fastest FSD system by more than an order of magnitude without sacrificing accuracy. The presented experiments showed that in contrast to UMass and LSH-FSD, two state-of-the-art FSD systems, kterm hashing operates in absolute constant time and space, independent of the number of documents processed. Second, this thesis was the first to show that the average novelty score of FSD systems decays over time. Additionally, we provided methods to counteract novelty decay and increase detection accuracy. Third, we presented an alternative application of kterm hashing by providing the first method to detect rumours in fast-moving streams in real time.

The introduction to this thesis outlined that historically, the advantage was with those who have access to information. The emergence of the internet and social media grants every one access to information about ongoing events world wide in real time. The advantage has shifted from those who have access to information to those who do so first. FSD is an active research field for more than 16 years. However, all recent development focused on improving the traditional comparison based methods. This thesis outlined the advantages and shortcomings of comparison based methods and introduced a new algorithm family able to maintain the high accuracy of comparison based methods while vastly improving their throughput. To the best of our knowledge FSD using kterm hashing reaches the highest reported throughput to this date.

While the primary focus of this thesis lied on detecting new events in data streams, we also provided evidence that kterm hashing is applicable beyond novelty detection. When detecting rumours, we applied kterm hashing as a real time textual entailment method that verified, whether the information of a tweet is confirmed by a set of news articles.

This points at future research and applications of kterm hashing. Additionally, we are interested in parametrising kterms by learning weights based on their attributes. Our work in rumour detection hints that kterms can be categorised by more than just cardinality to increase detection accuracy. We also ask ourselves how additional hashing functions and multiple memories would affect detection accuracy of kterm hashing. A different direction could be looking at parallelizing kterm hashing to further increase its throughput.

Bibliography

[1] Aggarwal Charu C . (2006). On biased reservoir sampling in the presence of stream evolution. In Proceedings of the 32nd international conference on Very large data bases, pages 607–618. VLDB Endowment.

[2] Allan J. (2002). Topic detection and tracking: event-based information organization. Kluwer Academic Publishers.

[3] Allan J., Lavrenko V., and Jin H. (2000a). First story detection in TDT is hard. In Proceedings of The 21st ACM International Conference on Information and Knowledge Management, pages 374-381. ACM.

[4] Allan J., Lavrenko V., Malin D., and Swan R. (2000b). Detections, bounds, and timelines: UMass and TDT-3. In Proceedings of Topic Detection and Tracking Workshop, pages 167-174.

[5] Allan J., Yang Y., Carbonell J., Yamron J., Doddington G., and Wayne C. (1998). TDT pilot study corpus. Catalog no. LDC98T25.

[6] Andoni A. and Indyk P. (2008). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Communications of the ACM.

[7] Austin A. (2008). MurmurHash. google.com/site/murmurhash. (last visited 12th March 2016).

[8] Jean-Philippe A. and Bernstein D. J. (2012). "SipHash" a fast short-input PRF.

[9] Becker H., Iter D., Naaman M., and Gravano L. (2012). Identifying content for planned events across social media sites. In Proceedings of the 5th ACM international conference on Web search and data mining, pages 533-542. ACM.

[10] Becker H., Naaman M., and Gravano L. (2011a). Beyond trending topics: Real- world event identification on Twitter. In Proceedings of the 5th International Conference on Weblogs and Social Media, pages 438-441. The AAAI Press.

[11] Becker H., Naaman M., and Gravano L. (2011b). Selecting quality Twitter content for events. In Proceedings of the 5th International Conference on Weblogs and Social Media, pages 442-445. The AAAI Press.

[12] Berinde R., Cormode G., Indyk P., Strauss M. J. (2009). Space-optimal heavy hitters with strong error bounds. Symposium on Principles of Database Systems.

[13] Bloom B. H. (1970). Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 13(7), 422-426.

[14] Bose P., Guo H., Kranakis E., Maheshwari A., Morin P., Morrison J. and Tang Y. (2008). On the false-positive rate of Bloom filters. Information Processing Letters, 108(4), 210-213.

[15] Brants T., Chen F., and Farahat A. (2003). A system for new event detection. In Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval, pages 330-337. ACM.

[16] Braun R. K. and Kaneshiro R. (2004). Exploiting topic pragmatics for new event detection in TDT-2004. Technical report, National Institute of Standards and Technology.

[17] Braverman V., Ostrovsky R., and Zaniolo C. (2012). “Optimal sampling from sliding windows.” Journal of Computer and System Sciences.

[18] Cai G., Wu H., Lv R. (2014). Rumours Detection in Chinese via Crowd Responses, ASONAM 2014, Beijing, China.

[19] Callison-Burch C. (2008). Syntactic constraints on paraphrases extracted from parallel corpora. In Proceedings of the Conference on Empirical Methods in Natural Language Processing.

[20] Castillo C, Mendoza M, Poblete B. (2011). Information Credibility On Twitter[C], The 20th International Conference on World Wide Web, Hyderabad, India,

[21] Cataldi M., Caro L. D., and Schifanella C. (2010). Emerging topic detection on Twitter based on temporal and social terms evaluation. In Proceedings of the 10th International Workshop on Multimedia Data Mining. ACM.

[22] Chang C. H. and Lin C. (2011). LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology.

[23] Charikar M. S. (2002). Similarity estimation techniques from rounding algorithms. In Proceedings of the 34th annual ACM symposium on Theory of computing, pages 380-388. ACM.

[24] Cordeir, M. (2012). Twitter event detection: Combining wavelet analysis and topic inference summarization. In Doctoral Symposium in Informatics Engineering, pages 123-138.

[25] Cortes C. and Vapnik V. (1995). Support-vector networks. Machine learning.

[26] Diaz F. and Metzler D. (2006). Improving the estimation of relevance models using large external corpora. In Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, pages 154-161. ACM.

[27] Fan L., Cao P., Almeida J. M. and Broder A. (2000). Summary cache: a scalable wide-area web cache sharing protocol. IEEEACM Transactions on Networking, 8(3), 281-293.

[28] Fiscus J. G. and Doddington G. R. (2002). Topic detection and tracking evaluation overview. Topic detection and tracking: event-based information organization, pages 17-31.

[29] Foo S. and Li H. (2004). Chinese word segmentation and its effect on information retrieval. Information processing and management, 40(1), pp.161-190.

[30] Genc Y., Sakamoto Y., and Nickerson J. V. (2011). Discovering context: Classifying tweets through a semantic transform based on Wikipedia. Foundations of Augmented Cognition. Directing the Future of Adaptive Systems.

[31] Gionis A., Indyk P. and Motwani R. (1999). Similarity search in high dimensions via hashing. In Proceedings of the 25th International Conference on Very Large Data Bases, pages 518-529. Morgan Kaufmann Publishers Inc.

[32] Sasa H., Ganitkevitch J., Ney H., and Andres-Ferrer J. (2008). "Triplet lexicon models for statistical machine translation." In Proceedings of the Conference on Empirical Methods in Natural Language Processing, pp. 372-381. Association for Computational Linguistics.

[33] Hu M., Sun A., and Lim E.-P. (2008). Event detection with common user interests. In Proceedings of the 10th ACM workshop on Web information and data management. ACM.

[34] Indyk P. and Motwani R. (1998). Approximate nearest neighbors: Towards removing the curse of dimensionality. In Proceedings of the 30th annual ACM symposium on Theory of computing, pages 604-613. ACM.

[35] Jurgens D. and Stevens K. (2009). Event detection in blogs using temporal random indexing. In Proceedings of the Workshop on Events in Emerging Text Types. Association for Computational Linguistics.

[36] Kriegel H.-P., Kroeger P. and Zimek, A. (2009). Outlier detection techniques. Tutorial at the 13th Pacific-Asia Conference on Knowledge Discovery and Data Mining.

[37] Krovetz R. (1993). Viewing morphology as an inference process. In Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval, pages 191-202. ACM.

[38] Kumaran G. and Allan J. (2005). Using names and topics for new event detection. In Proceedings of the conference on Human Language Tech-

nology and Empirical Methods in Natural Language Processing, pages 121-128. Association for Computational Linguistics.

[39] Kwak H., Lee C., Park H. and Moon, S. (2010). What is Twitter, a social network or a news media? In Proceedings of the 19th International Conference on World Wide Web, pages 591-600. ACM.

[40] Kwon S., Cha M., Jung K., Chen W. and Wang Y. (2013). "Prominent features of rumor propagation in online social media", Data Mining (ICDM), IEEE.

[41] Lavrenko V. (2004). A generative theory of relevance. PhD thesis, University of Massachusetts.

[42] Lavrenko V., Allan J., DeGuzman E., LaFlamme D., Pollard V., and Thomas S. (2002). Relevance models for topic detection and tracking. In Proceedings of the 2nd international conference on Human Language Technology Research, pages 115-121. Morgan Kaufmann Publishers Inc.

[43] Li H., Chen H. (2008) A Deleting Strategy in Mining Frequent Itemsets over Sliding Window of Stream. IEEE.

[44] Li C., Sun A., and Datta A. (2012a). Twevent: Segment-based event detection from tweets. In Proceedings of ACM Conference on Information and Knowledge Management. ACM.

[45] Li R., Lei K. H., Khadiwala R., and Chang K. C. C. (2012b). TEDAS: A Twitter- based event detection and analysis system. In Proceedings of 28th International Conference on Data Engineering, IEEE Computer Society.

[46] Blerina L., Kolomvatsos K., and Hadjiefthymiades S. (2014). "Facing the cold start problem in recommender systems." Expert Systems With Applications 41.4.

[47] Liu X., Nourbakhsh A., Li Q., Fang R. and Shah S. (2015). Real-time rumor debunking on twitter, in Proceedings of the 24th ACM International Conference on Information and Knowledge Management. ACM.

[48] Luo G., Tang C. and Yu, P. S. (2007). Resource-adaptive real-time new event detection. In Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pages 497-508. ACM.

[49] Madnani N. and Dorr B. (2010). Generating phrasal and sentential paraphrases: A survey of data-driven methods. Computational Linguistics, 341:387.

[50] Mathioudakis M. and Koudas N. (2010). Twittermonitor: Trend detection over the Twitter stream. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pages 1155-1158. ACM.

[51] McCreadie R., Macdonald C., Ounis I., Petrovic S., Osborne M. (2013). Scalable Distributed Event Detection for Twitter. IEEE International Conference on Big Data. Santa Clara US.

[52] Mendoza M., Poblete B., Castillo C. (2010). Twitter Under Crisis: Can we Trust What we RT?, The 1st Workshop on Social Media Analytics, SOMA.

[53] Metzler D., and Croft B. (2005) "A Markov random field model for term dependencies." In Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 472-479. ACM.

[54] Metzler D., Cai C., and Hovy E. (2012). Structured event retrieval over microblog archives. In Proceedings of Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, pages 646-655. Association for Computational Linguistics.

[55] Muthukrishnan S. M. (2005). Data streams: algorithms and applications. Foundations and Trends in Theoretical Computer Science, 1(2):117:236.

[56] O'Connor B., Balasubramanyan R., Routledge B. R. and Smith N. A. (2010). From tweets to polls: Linking text sentiment to public opinion

time series. In Proceedings of the 4th International Conference on Weblogs and Social Media, pages 122-129. The AAAI Press.

[57] Osborne M., Petrovic S., McCreadie R., Macdonald C. and Ounis I. (2012). Bieber no more: First story detection using Twitter and Wikipedia. In Proceedings of the SIGIR workshop on Time-Aware Information Access.

[58] Osborne M. and Dredze M. (2014). Facebook, Twitter and Google Plus for Breaking News: Is there a winner?. ICWSM, Ann Arbor USA.

[59] Osborne M., Lall A. and Van Durme B. (2014). Exponential reservoir sampling for streaming language models. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 687–692. Association for Computational Linguistics.

[60] Ozdikis O., Senkul P. and Oguztuzun H. (2012). Semantic expansion of hashtags for enhanced event detection in Twitter. In Proceedings of the 1st International Workshop on Online Social Systems.

[61] Papka R., Allan J., and Lavrenko V. (1999). UMass approaches to detection and tracking at TDT2. In DARPA: Broadcast News Workshop, pages 111-116.

[62] Petrovic S., Osborne M. and Lavrenko V. (2010). Streaming first story detection with application to Twitter. In Proceedings of the 11th annual conference of the North American Chapter of the Association for Computational Linguistics

[63] Petrovic S., Osborne M. and Lavrenko V. (2012). Using paraphrases for improving first story detection in news and Twitter. In Proceedings of Human Language Tech- nologies: Conference of the North American Chapter of the Association for Computational Linguistics, pages 338-346. Association for Computational Linguistics.

[64] Petrovic S. (2012). Real-time Event Detection in Massive Streams. PhD Thesis.

[65] Petrovic S., Osborne M., McCreadie R., Macdonald C., Ounis I., Shrimpton L. (2013) Can Twitter replace Newswire for breaking news?. ICWSM, Boston US.

[66] Phuvipadawat S. and Murata T. (2010). Breaking news detection and tracking in Twitter. In Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, pages 120-123. IEEE Computer Society.

[67] Pike G. and Alakuijala J. (2011). The CityHash family of hash functions.

[68] Popescu A.-M. and Pennacchiotti M. (2010). Detecting controversial events from Twitter. In Proceedings of the 19th ACM international conference on Information and knowledge management. ACM.

[69] Qazvinian V., Rosengren E., Radev D. R., Mei Q. (2011) Rumour has it: Identifying Misinformation in Microblogs, EMNLP, Edinburgh, UK.

[70] Qin Y.,Wurzer D., Lavrenko V. and Tang C. (2017). "Counteracting Novelty Decay in First Story Detection." In ECIR - European Conference on Information Retrieval, pp. 555-560. wode airen Springer.

[71] Richardson M, Agrawal R, Domingos P. (2003). Trust Management for the Semantic Web. The Semantic Web-ISWC 2003, Heidelberg: Springer, Berlin - Heidelberg.

[72] Levy R. and Manning C. D. (2003). Is it harder to parse Chinese, or the Chinese Treebank?. ACL 2003.

[73] Sakaki T., Okazaki M., and Matsuo Y. (2010). Earthquake shakes Twitter users: real-time event detection by social sensors. In Proceedings of the 19th International Conference on World Wide Web, pages 851-860. ACM.

[74] Sankaranarayanan J., Samet H., Teitler B. E., Lieberman M. D., and Sperling J. (2009). Twitterstand: news in tweets. In Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM.

[75] Shrimpton L., Lavrenko V., Osborne M. (2015). Sampling Techniques for Streaming Cross Document Coreference Resolution. Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the ACL.

[76] Subasic I. and Berendt B. (2011). Peddling or creating? Investigating the role of Twitter in news reporting. Advances in Information Retrieval.

[77] SunS., Liu H., He J., Du X. (2013). Detecting Event rumours on Sina Weibo Automatically, APWeb.

[78] TDT by NIST (1998-2004). http://www.itl.nist.gov/iad/mig/tests/tdt/resources.html (Last Update: 2008)

[79] Tong X., Zai C., Milic-Frayling C., and Evans D. A. (1996). Experiments on Chinese text indexing – CLARIT TREC–5 Chinese track report. http://trec.nist.govpubs trec5t5_ proceedings.html, Maryland

[80] Tseng H., Chang P., Andrew G., Jurafsky D. and Manning C. (2005). A Conditional Random Field Word Segmenter. In Fourth SIGHAN Workshop on Chinese Language Processing.

[81] Vitter J. S. (1985). Random sampling with a reservoir. ACM Transactions on Mathematical Software (TOMS), 11(1):37–57.

[82] Weng J., Yao Y., Leonardi E., and Lee F. (2011). Event detection in Twitter. In Proceedings of the 5th International Conference on Weblogs and Social Media, . The AAAI Press.

[83] Wong S. K. M., Ziarko W., and Wong P. C. N. (1985). Generalized vector spaces model in information retrieval. In Proceedings of the 8th annual international ACM SIGIR conference on Research and development in information retrieval. ACM.

[84] Wang S., Terano T. (2015). Detecting rumour patterns in streaming social media, Guimi, IEEE.

[85] Wu K., Yang S., Zhu K. (2015). False rumours Detection on Sina Weibo by Propagation Structures, In the Proceedings of ICDE.

[86] Wurzer D., Lavrenko V., Osborne M. (2015). Tracking unbounded Topic Streams. In the Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics, ACL.

[87] Wurzer D., Lavrenko V., Osborne M. (2015). Twitter-scale New Event Detection via K-term Hashing. In the Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP.

[88] Yang Y., Pierce T. and Carbonell J. (1998). A study of retrospective and on-line event detection. In Proceedings of the 21st annual international ACM SIGIR con- ference on Research and development in information retrieval. ACM.

[89] Yang F., Yu X., Liu Y., Yang M. (2012). Automatic Detection of Rumor on Sina Weibo. MDS'12.

[90] Yardi S., Romero D., Schoenebeck G., and Boyd D. (2009). Detecting spam in a Twitter network. First Monday, 15(1).

[91] Zeng J., Gong L., Wang Q., Wu. C. (2009). Hierarchical for Topic Analysis Based on Variable Feature Reduction. IEEE.

[92] Zhang J., Ghahramani Z. and Yang Y. (2005). A probabilistic model for online document clustering with application to novelty detection. In Advances in Neural Information Processing Systems 17.

[93] Zhao Z., Resnick P. and Mei Q. (2015). Enquiring Minds: Early Detection of Rumors in Social Media from Enquiry Posts. In the Proceedings of WWW.

[94] Zhou X., Cao J., Jin Z., Fei X., Su Y., Zhang J., Chu D. and Cao X. (2015). Realtime news certification system on sina weibo. WWW.